ESCOLA POLITÉCNICA
DE PERNAMBUCO

Análise de Energia, Desempenho e Área para Hierarquia de Memória com dois níveis

Trabalho de Conclusão de Curso

Engenharia da Computação

Washington Luís Soares dos Santos
Orientador: Prof. Abel Guilhermino da Silva Filho

Recife, Junho de 2008

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Análise de Energia, Desempenho e Área para Hierarquia de Memória com dois níveis

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Washington Luís Soares dos Santos
Orientador: Prof. Abel Guilhermino da Silva Filho

Recife, Junho de 2008

Washington Luís Soares dos Santos

Análise de Energia, Desempenho e Área para Hierarquia de Memória com dois níveis

Resumo

Com a rapidez com que a tecnologia se desenvolve atualmente, e com a alta concorrência entre fabricantes de processadores e sistemas embarcados, se faz extremamente necessário a utilização de arquiteturas para construção desses equipamentos de forma que os custos possam ser otimizados, ajustando-se a arquitetura a uma determinada aplicação, através de simuladores de arquiteturas, não se esquecendo de atender as restrições impostas pelo sistema tais como: área, desempenho e consumo de energia. Uma vez que seja possível chegarmos as melhores configurações para uma determinada arquitetura antes de implementá-las, os fabricantes estarão sempre um passo a frente de seus concorrentes atendendo conseqüentemente a janela de mercado.

Um componente de grande importância nessa conjectura é a memória cache, que vem agregando grande valor de desempenho aos hardwares atuais, mas que devem ser modeladas de forma consciente, pois juntamente com o aumento de desempenho, vem o aumento no custo e no consumo de energia.

Este trabalho visa o estudo de desempenho, consumo de energia e área de aplicações específicas para sistemas embarcados para uma hierarquia de memória com dois níveis de cache, sendo o segundo nível unificado. Será avaliado o impacto em termos destas três restrições não funcionais considerando variação em alguns parâmetros da hierarquia e alguns *benchmarks* para avaliação de resultados obtidos dos simuladores de arquitetura e modelo de energia e área para memória cache.

Abstract

With the speed at which technology is developed today, and the high competition among manufacturers, processors and embedded systems, is extremely necessary the use of architectures for construction of the equipment so that costs can be optimized, setting up the architecture for a specific application, through simulators of architectures, Without forgetting to take the restrictions imposed by the system such as: area, performance and power consumption. Since it is possible to get the best settings for a specific architecture before implementing them, manufacturers are one step ahead of its competitors consequently given the window of market.

A major component of this conjecture is the cache memory, that is adding great value to the performance of current hardware, but they must be shaped so aware, because along with the increase in performance, is the increase in cost and in energy consumption.

This work aims to study the performance, power consumption and area-specific applications for systems embeded into a hierarchy of memory with two levels of cache, with the second level unified. The impact will be assessed in terms of these three functional restrictions not considering changes in some parameters of the hierarchy and some benchmarks for assessing the results of simulators and architecture model of energy and area to cache.

.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

CPU – *Central Processing Unit*(Unidade Central de Processamento)

RAM – *Random Access Memory*(Memória de Acesso Randômico)

SRAM – *Static RAM*(RAM Estática)

DRAM – *Dynamic RAM*(RAM Dinâmica)

HD – *Hard Disc*(Disco Rígido)

KByte – KiloByte= 1024 Bytes

MByte – MegaByte = 1024 Kbytes

ROM – *Read Only Memory*(Memória Apenas de Leitura)

PROM – *Programed ROM*(ROM Programável)

EPROM – *Erasable PROM*(PROM Apagável através de luz Ultravioleta)

EEPROM – *Eletrical Erasable PROM* (PROM Apagável por Eletricidade)

SDRAM – *Synchronous Dynamic RAM* (RAM dinâmica síncrona)

DDR – Double Data Rate(Taxa Duplicada de transferência de Dados)

LRU – *Least Recently used*(Último Recentemente Usado)

FIFO – *First-In-First-Out*(Primeiro a entrar – Primeiro a sair)

LFU – *Least Frequently used*(Último Frequentemente Usado)

L1 – Memória Cache de Nível 1

L2 – Memória Cache de Nível 2

ADL – *Architecture Design Language*(Linguagem de Descrição de Arquitetura)

SoC – *System On Chip*(sistema dentro de um chip)

LD1 – Memória de Dados da Cache de Nível 1(= DL1)

LI1 – Memória de Instruções da Cache de Nível 1 (= IL1)

LD2 – Memória de Dados da Cache de Nível 2(= DL2)

LI2 – Memória de Instruções da Cache de Nível 2 (= IL2)

Agradecimentos

Agradeço primeiramente a Deus e a Jesus Cristo por esta oportunidade de conclusão do curso de engenharia, por tudo que tenho, que alcancei e que sei que alcançarei.

A minha família, em especial aos meus pais e irmã pelo apoio, pela compreensão, amor e carinho que sempre me deram. Amo-os muito.

A minha noiva Edna Carla, futura esposa, a quem amo muito, pelo apoio, compreensão, amor e companheirismo.

Aos meus grandes amigos de todas as horas de minha vida, de longas datas, que graças a eles, também, me encontro aqui hoje, que Deus os abençoe sempre, cada vez mais.

Ao professor Abel Guilhermino, meu orientador, pela paciência, compreensão, sabedoria e integridade, pois sei que não foi fácil me orientar.

A todos os professores do DSC da UPE, verdadeiros guerreiros que construíram esse departamento, do qual nos orgulhamos, com muita seriedade, dignidade e sinceridade. Aprendo cada vez mais com os senhores.

Aos meus amigos de trabalhos, meus chefes, em especial a Raul César, Henrique e Meira, pela compreensão em todos os momentos que foram necessários para a conclusão deste trabalho.

Que Deus abençoe cada um de vocês, iluminando e abençoando tudo que fizerem com amor, em beneficio do próximo. Muita Paz, Saúde e Sucesso a todos.

Que possamos estar sempre unidos, por mais que a correria da vida nos afaste fisicamente.

Capítulo 1

Introdução

Com o avanço tecnológico e a crescente necessidade de poder de processamento na manipulação das informações e na geração de resultados, as empresas estão investindo cada vez mais na criação de hardwares menores e com maior capacidade de processamento.

Uma vez que os processadores estão cada vez mais rápidos e que os demais componentes de um sistema não acompanham esta velocidade, se faz necessário a implementação de soluções que auxiliem no aumento da velocidade de resposta para as tarefas a serem executadas, para que o processador não precise esperar muito por elas.

Tendo esse problema em vista, fez-se uso do desenvolvimento de uma memória mais rápida do que a memória principal que trabalha a uma velocidade próxima a do processador, porém um pouco mais lenta do que os registradores que compõem a CPU (Unidade Central de Processamento) juntamente com o processador[1]. Essa memória, chamada de memória cache, apesar de ser mais eficiente (tecnologia SRAM – Memória Estática de Acesso Aleatório), também é mais cara se comparada à tradicional DRAM (Memória Dinâmica de Acesso Aleatório), sendo objeto de análise o tamanho a ser incorporado ao chip dos equipamentos, para que o custo não se sobreponha ao desempenho. Com ela o processador não precisa acessar o barramento externo para buscar instruções e dados, aumentando consideravelmente o seu desempenho[1][2].

Atualmente já encontram-se memórias cache em HD's (*Hard Disk* – Disco Rígido) e placas mãe, dentre outros equipamentos, comprovando, assim, a eficiência no tempo de resposta de processamento desses [2].

Sendo mais comum nas CPU de computadores, essas memórias podem ser construídas em diversas arquiteturas e em mais de um nível, aumentando ainda mais o desempenho dos computadores e de sistemas embarcados. Sistemas estes que se utilizam do processador e do acesso a memória nas soluções específicas para um determinado problema, já que o computador é de uso genérico para diversas aplicações.

Segundo Patterson [4] o custo de 1 MByte de memória era de U$5,00(cinco dólares), isso em 1997, mas, informações atuais indicam que o custo de 1MByte de memória cache fica em torno de U$15,00(quinze dólares) a U$20,00(vinte dólares) [5], portanto, para cálculos futuros será utilizado o valor de U$20,00(vinte dólares). Com isso, percebe-se a importância de um estudo maior, incluindo a aplicação de simuladores e de heurísticas, da melhor configuração de cache antes de implementá-la no *hardware* a ser construído, pois quanto maior o tamanho da cache, maior o seu custo e, conseqüentemente, maior o valor do *hardware*.

1.1 Objetivos Gerais

O objetivo geral deste trabalho abrange uma série de aspectos, dentre eles pode-se citar: o uso de simuladores de arquitetura e de modelo de memória cache visando realizar análises em termos de consumo de energia, desempenho e custo com base em aplicações direcionadas para sistemas embarcados. Através desse estudo, será possível determinarmos uma relação entre esses três parâmetros em arquiteturas de memórias cache. Isso possibilitará, a implementação e fabricação de componentes que se utilizarão de memórias cache de dois níveis com o segundo nível unificado, o que está se tornando uma tendência nos processadores atuais.

1.2 Objetivos Específicos

O objetivo principal deste projeto de final de curso visa à análise em termos de área, consumo de energia e desempenho de hierarquias de memória com dois níveis de cache e o segundo nível unificado considerando aplicações específicas para sistemas embarcados. Entre esses estudos estão o uso do simulador SimpleScalar, para obtenção

de informações de desempenho das aplicações, e o eCACTI, para extração de informações de consumo de energia e área de memória cache.

1.3 Estrutura do Trabalho

Este trabalho de conclusão de curso encontra-se divididos nos seguintes capítulos subseqüentes:

Capítulo 2 – Memórias Caches: serão expostos conceitos relacionados às memórias caches, suas possíveis arquiteturas e níveis existentes.

Capítulo 3 – Simuladores de memória Cache: onde será mostrado uma relação de possíveis simuladores e de *benchmarks*, e explicadas as ferramentas utilizadas para simular as arquiteturas de memórias cache, que darão como resultado o consumo de energia, ciclos e área.

Capítulo 4 – Metodologia Proposta: abordando os passos dos objetivos específicos deste trabalho, detalhando-os e demonstrando a importância de cada um deles.

Capítulo 5 – Estudo de Casos: nesse capítulo serão vistos a arquitetura do sistema e do hardware onde as configurações foram simuladas, chamado de ambiente experimental, assim como serão avaliados os resultados obtidos dessas simulações, fazendo a comparação entre aplicações diferentes, comparando os componentes de área, consumo de energia e desempenho da arquitetura a ser avaliada.

Capítulo 6 – Conclusão: onde serão expostas as conclusões finais do trabalho, fazendo uma análise crítica sobre o impacto de uma abordagem que considera as 3 componentes (área, energia e desempenho), e onde serão indicados trabalhos futuros para a continuidade deste.

Capítulo 2

Memória Cache

Iniciando os estudo sobre memórias cache é preciso obter primeiramente algumas definições básicas, sendo a primeira delas o conceito de memória.

Memória é todo local onde pode-se armazenar informações [6]. Elas são classificadas como voláteis, perdendo seu conteúdo quando retirada a energia que as alimenta, ou não voláteis, quando, mesmo sem energia para alimentá-las, os dados gravados nelas permanecem intactos.

As memórias podem ser do tipo:

- Memória RAM(*Random Access Memory*): são as memórias que armazenam dados e programas utilizados pelo processador no exato momento em que estão sendo executados. São de fácil manipulação, pois permitem a escrita e a leitura de forma simples e rápida. As memórias RAMs podem ser estáticas (SRAM) ou dinâmicas (DRAM). As primeiras são utilizadas na memória cache, sendo mais rápidas e mais caras e não necessitam ser recarregadas a cada momento (serão abordadas no tópico 2.3), e as dinâmicas, que são mais baratas e mais lentas do que as estáticas, pois necessitam de um circuito de *refresh*. Esse circuito tem por objetivo recarregar periodicamente as memórias DRAM, mantendo assim, as informações nela armazenadas [6][9].
- Memória de armazenamento: que são os dispositivos de armazenamento de dados não voláteis, como os Discos Rígidos do computador, os CD-ROMs e DVDs, por exemplo. Apesar de alguns desses dispositivos serem de grande capacidade de armazenamento, a gravação e a recuperação das informações se faz de forma muito mais lenta do que nas memórias RAM [6][7][8];

- Memória ROM(*Read Only Memory* – Memória apenas de Leitura): são memórias que também não são voláteis, mas que são utilizadas geralmente com o intuito de apenas leituras em sistemas computacionais, já vêm programadas de fabrica.
- Memória PROM *(Programed ROM* - ROM Programável): que permite apenas uma gravação pelo usuário final;
- Memória EPROM (*Erasable PROM* - PROM Apagável através de luz Ultra-violeta): permite ser apagada através de luz ultravioleta e regravada algumas vezes;
- Memória EEPROM (*Eletrical EPROM* - PROM Apagável por Eletricidade): tem o mesmo principio da memória EPROM, mas é apagada utilizando-se uma tensão elétrica em vez de luz ultravioleta [6][9];
- Memória FEPROM (*Flash EPROM*): Pode ser gravada e apagada várias vezes. Consomem menos energia que as EEPROM. Utilizada em *pendrive*, cartões de memória.

2.1 – Tecnologias de memórias RAM SIMM, DIMM e DDR1, 2 e 3.

Tendo o conhecimento de que o poder de processamento de um computador ou sistema embarcado não depende apenas do *clock* do seu processador, mas de outros componentes, as empresas resolveram investir no aumento da velocidade de processamento das memórias RAMs.

Esse avanço pode ser notado através da diferença da velocidade dessas memórias, onde destacaremos os tipos mais comuns entre elas:

- SDRAM: RAM dinâmica síncrona, também conhecida como DIMM(*Dual inline memory module*), são mais antigas e mais lentas[8]. Podem ser divididas em:
    - PC-100: trabalha a uma velocidade de 100MHz, tendo uma taxa de transferência de 800MBytes/s;

  - PC-133: trabalha a velocidade de 133MHz tendo uma taxa de transferência de 1066MBytes/s;

- DDR (*Double Data Rate*): foram criadas para se aproveitarem tanto da transição de subida quanto a de descida do *clock* de transferência de dados[8]. Podem ser classificadas em:
  - DDR-200(PC1600): trabalha a 200MHz, transferindo 1600MBytes/s de dados;
  - DDR-266(PC2100): trabalha a 266MHz, com uma taxa de transferência de 2100MBytes/s;
  - DDR-333(PC2700): trabalha a 333MHz, com taxa de transferência de 2700MBytes/s;
  - DDR-400(PC3200): trabalha a 400MHz com taxa de transferência de dados de 3200MBytes/s;
- DDR2: Mais rápidas do que a DDR, trabalhando com uma velocidade de transferência maior. São dos tipos:
  - DDR2-533(PC4200): trabalha a 533MHz, com taxa de transferência de 4200MBytes/s;
  - DDR2-667(PC5300): trabalha a 667MHz, com taxa de transferência de 5300MBytes/s;
  - DDR2-800(PC6400): trabalha a 800MHz, com taxa de transferência de 6400MBytes/s;
- DDR3: é a terceira geração das memórias DDR, muito mais rápidas. Tem a sua freqüência de trabalho que varia de 800MHz a 2000MHz, podendo chegar a taxas de 25.4 GBytes/s [8][10].

Apesar de toda a evolução sofrida pelas memórias RAM em um curto espaço de tempo, elas ainda causam ao processador uma ociosidade, devido à espera na busca das informações, pois os processadores atuais trabalham em freqüências na ordem de GHz. Faz-se necessário, então, o uso cada vez mais das memórias caches, diminuindo o acesso ao barramento principal e trabalhando, praticamente, na mesma velocidade do processador.

2.2 – Memórias Cache

De acordo com a lei de Moore, figura 2.1, a cada 18 meses a quantidade de transistores para uma mesma unidade de área dobraria. No entanto, observa-se que a velocidade das memórias RAMs não conseguiu acompanhar este desenvolvimento, gerando nos processadores um estado chamado de *wait state*, estado de espera. Nesse estado, o processador fica ocioso, esperando pelo resultado da operação que a memória RAM estava executando, para só então prosseguir na execução das aplicações [3].

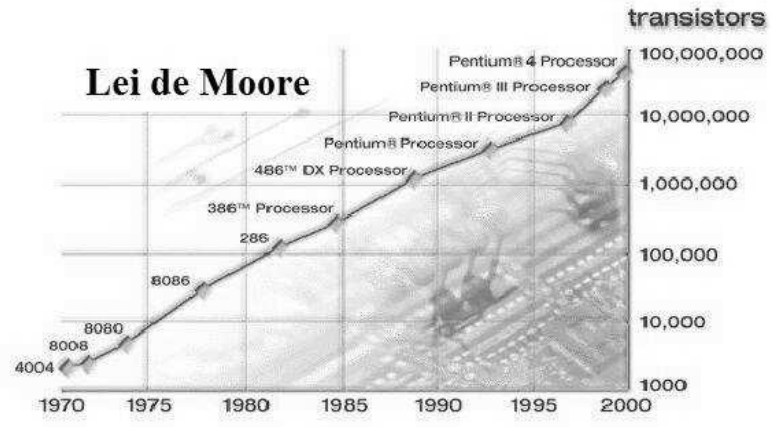

Figura 2.1: Lei de Moore(retirada de [3])

Foi então criada uma memória, chamada cache, de acesso intermediário entre os registradores e a memória RAM, com tecnologia SRAM, que é muito mais rápida do que o acesso as memórias DRAMs, e um pouco mais lentas do que os registradores. Essa memória possui menor capacidade do que a memória principal, e normalmente encontra-se implementada dentro da mesma pastilha de semicondutor do processador. Dessa forma, o processador consegue acessar as informações na memória cache praticamente na mesma velocidade em que trabalha, permitindo que o sistema ganhe desempenho como um todo[3]. Será visto mais detalhadamente na seção 2.2.1 o funcionamento delas.

Na figura 2.2 observa-se a hierarquia de memórias, levando-se em conta o tempo de acesso e o custo delas.

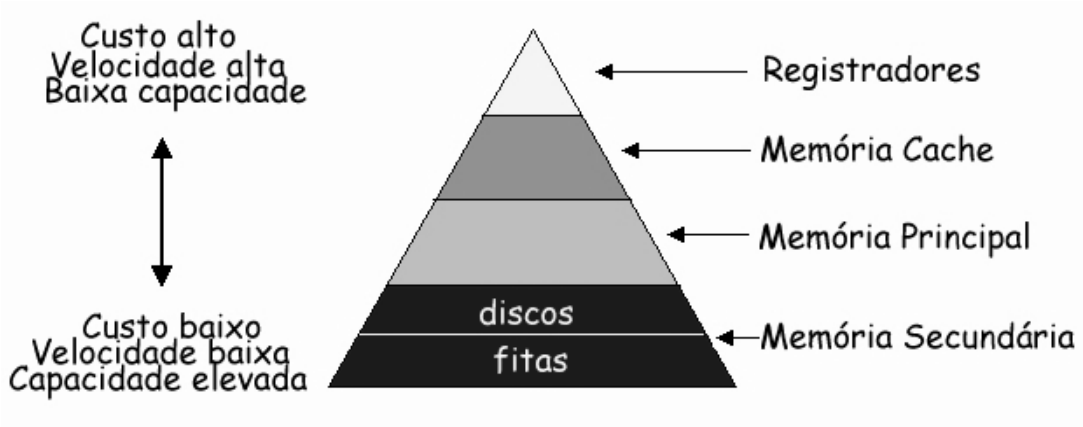

Figura 2.2: Hierarquia de memórias, retirada de [10].

Na figura 2.3, modificada de [7], pode-se verificar uma pequena ilustração da organização de um computador que possui o circuito de memória cache.

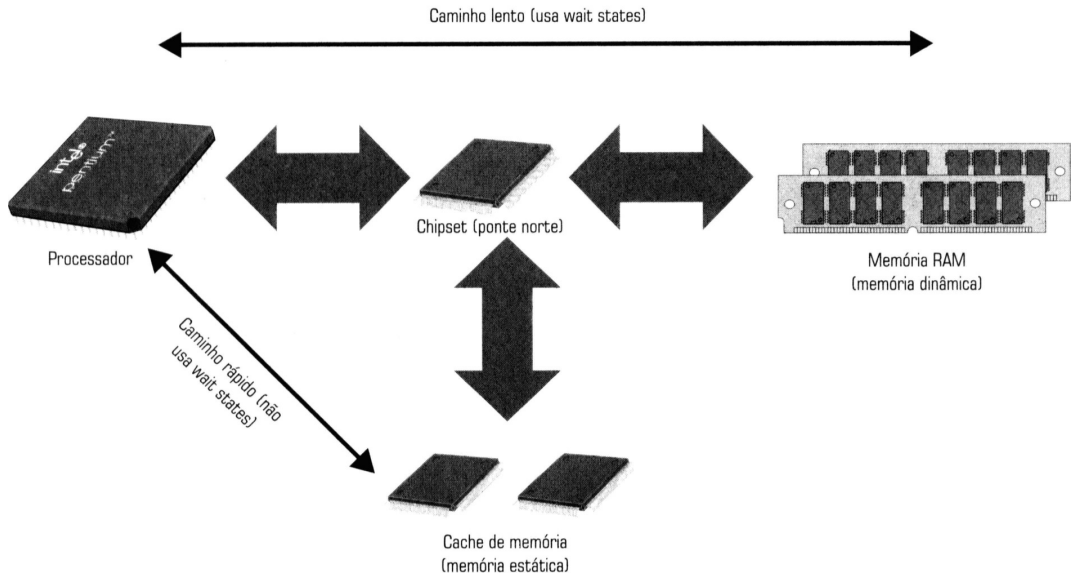

Figura 2.3: Ilustração de comunicação entre processador e memórias

Os processadores podem conter mais de um nível de memória cache. Nos modelos iniciais de sua utilização, as memórias eram instaladas externamente ao processador, mas, nas versões mais modernas, encontram-se dentro deles [7]. Nas figuras 2.4 e 2.5, pode-se ver uma ilustração da comunicação entre processador e memórias, com apenas 1 nível de cache (L1) interna no processador, figura 2.4, e com 2 níveis de cache (L1 e L2) internas, figura 2.5.

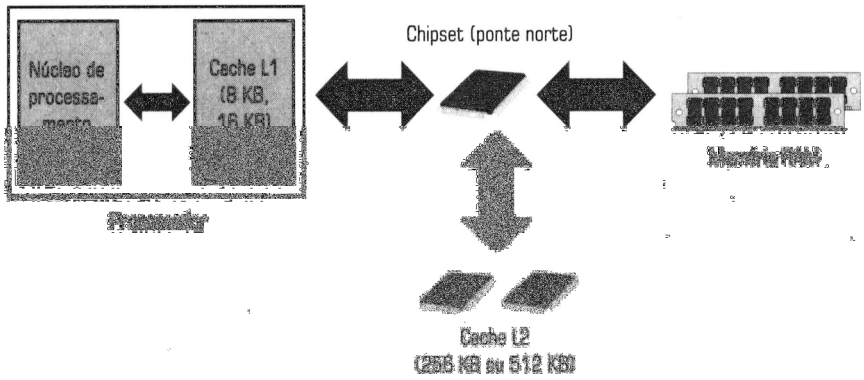

Figura 2.4: Ilustração de comunicação de processador com cache L1 interna.

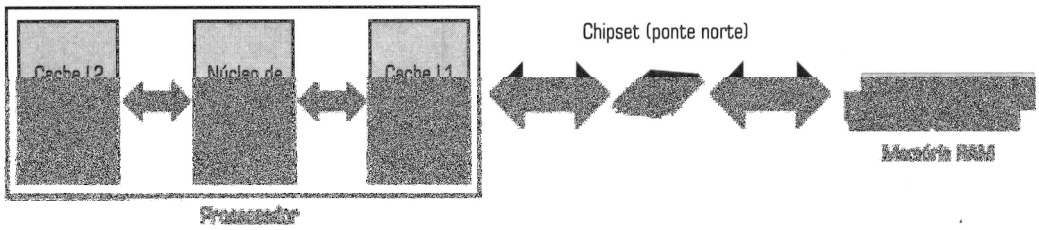

Figura 2.5: Ilustração de comunicação de processador com cache L1 e L2 internas.

2.2.1 – Funcionamento de memórias cache, fluxo de dados

A memória cache utiliza-se de um mapeamento de informações para saber quais dados da memória principal devem ser nela armazenados. Este mapeamento se dá pelos princípios de [3][4]:

1 – Principio da Localidade Temporal: se um item é acessado, a probabilidade dele ser acessado novamente é muito grande, dentro de um espaço curto de tempo;

2 – Princípio da Localidade Espacial: se um item é acessado, os itens próximos a ele têm uma alta probabilidade de serem acessados.

Seguindo essa linha de raciocínio, existem 3 técnicas pelas quais a memória cache copia estes itens acessados recentemente da memória principal[3][4]:

1 – Mapeamento Direto: cada bloco da memória principal é mapeado em uma única linha de cache. Esta técnica tem a vantagem de ser de simples implementação, porém possui uma grande desvantagem, é que, como varias linhas da memória principal são referenciados por apenas uma linha na cache, se o processador utilizar informações contidas em endereços diferentes na memória principal e a cache referenciar estes endereços em uma única linha, ela terá que ficar trocando este dado muitas vezes, podendo perder desempenho com este procedimento para determinadas aplicações;

2 – Mapeamento Associativo: este tipo de mapeamento vem a minimizar a desvantagem do anterior. Nesse caso os dados podem ser carregados em qualquer posição da cache, mesmo pertencendo ao mesmo bloco de informações, o que não ocorria no mapeamento direto. A desvantagem é que, para que isso seja possível, o controlador precisa comparar os rótulos dos endereços solicitados pelo processador, para verificar se os dados se encontram na cache. É uma tarefa complexa e que demanda tempo, pois será pesquisada em toda a cache;

3 – Mapeamento Associativo por Conjunto: esta técnica junta as vantagens das duas abordagens anteriores, minimizando suas desvantagens. Este mapeamento divide a cache em blocos que podem conter endereços de um único bloco da memória principal, diferente do mapeamento direto que utilizava apenas uma linha para mapear blocos, diminuindo o tamanho da pesquisa pelo controlador para verificar a existência, ou não, da informação requerida.

Nesse ponto surge uma pequena pergunta, como o controlador sabe que informação pode substituir na cache por novas informações vindas da memória principal? Quando a cache vai ser atualizada, ela se utiliza de um algoritmo de substituição, sendo os mais conhecidos os que seguem abaixo [3][4][9]:

1 – Menos Usado Recentemente (LRU – *Less Recentily Used*): substituindo, como o próprio nome diz, os blocos que não foram acessados a um certo tempo;

2 – Primeiro a Entrar, Primeiro a Sair (FIFO – *First In, First Out*): este algoritmo substitui o bloco que se encontra a mais tempo copiado na cache, independente se ele esta sendo acessado;

3 – Menos Freqüentemente Usado (LFU – *Less Frequentily Used*): Este substitui o bloco que foi menos acessado na cache.

Vale a pena salientar que o controlador da memória deve se preocupar também com a atualização dos dados, garantindo que os dados que se encontram na cache estão ou pelo menos, ao serem substituídos, estarão iguais aos dados que se encontram na memória principal. Atualizações estas que podem ser do tipo escrita direta (*write-through*), onde, ao se atualizar uma informação na memória cache, esta mesma informação é atualizada na memória principal, ou do tipo escrita de volta (*write-back*) que ocorre quando o bloco da cache vai ser substituído e este, contendo o *flag* indicador de atualização ativado, será copiado na memória principal, atualizando a informação[2][4].

Tendo como base a ilustração mostrada na figura 2.4, nas configurações onde as caches possuíam apenas um único nível dentro do processador, o funcionamento da mesma procede de acordo com a ilustração simplificada da figura 2.6, onde, quando o processador necessita buscar dados para a execução de uma tarefa, ele busca primeiramente na memória cache mais próxima a ele, a cache L1. Se essa informação não estiver presente na L1, o controlador de memória terá que procurar este dado na memória principal através do acesso ao barramento, conseqüentemente demandando mais tempo para a obtenção da mesma. Esta informação, ao ser encontrada, é gravada na memória cache e levada também para o processador para que ele continue a execução da tarefa.

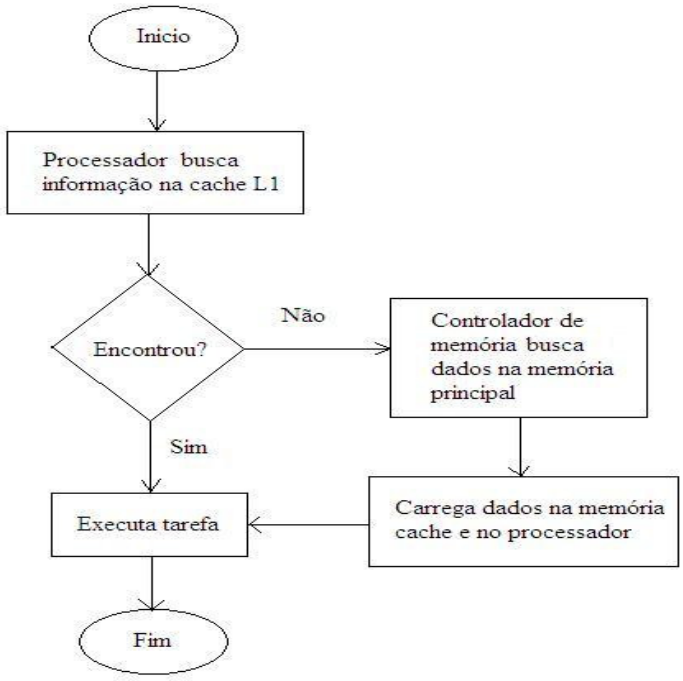

Figura 2.6: Ilustração de fluxo de dados para cache de um nível (L1)

Já na figura 2.5, tem-se uma organização onde o processador possui dois níveis de cache, a L1 e a L2 internas, onde o processador, ao buscar uma informação na cache L1, se esta informação não se encontra nesse nível, a cache L2 será consultada, o que também acarretará em um atraso na recuperação da informação. E se ocorrer o caso onde a informação não for encontrada na L2, o controlador terá que realizar a procura na memória principal, acarretando um maior atraso.

Quando o processador busca uma informação na cache e a encontra, diz-se que ocorreu um acerto (*HIT)* na cache, e quando essa busca não encontra a informação desejada, dizemos que ocorreu um erro (*MISS)* na cache[3][9].

2.2.2 – Arquiteturas de níveis de cache

Como visto no tópico anterior, podem existir mais de um nível de cache nos processadores, na verdade alguns processadores atuais já se utilizam de 4 níveis de cache distribuídos de acordo com a arquitetura de cada um deles, sendo este último nível geralmente localizado na placa mãe. Níveis esses que, em conjunto com técnicas como o *pipeline*, aumentam consideravelmente o desempenho do sistema[2].

Deve-se atentar a uma pequena observação, as caches de nível superior devem ter seu tamanho maior do que tamanho da cache imediatamente anterior a ela, isto é, a cache L2, por exemplo, deve ser maior do que a cache L1, para que sua funcionalidade seja alcançada, pois os níveis de cache nada mais são do que uma filtragem dos dados pertencentes à memória imediatamente superior a elas, deixando os dados cada vez mais próximos do processador, aumentando assim a velocidade de acesso aos mesmos [3].

Cada nível de cache pode ser distribuído de duas formas: separado ou unificado. Cache com níveis separados significa que a cache possui dois circuitos separados, um utilizado para armazenar apenas instruções e outro utilizado para armazenar apenas dados, também chamadas de cache de instruções e cache de dados, vinculadas ao nível pertencentes. A grande vantagem desse tipo de abordagem é que os acessos a instruções e dados podem ser feitos de forma paralela, aumentando o desempenho do processamento. A desvantagem é que, por ser dividida, ela tem uma capacidade limitada para dados e para instruções.

Já na cache com nível unificado não existe essa separação entre dados e instruções, tendo a vantagem de poder copiar uma quantidade maior de dados do que de instruções, de acordo com a demanda do processador. Mas tem a desvantagem de não permitir o acesso paralelo a dados e a instruções, o acesso passa a ser seqüencial, tendo, o processador, que esperar se estiver precisando de instruções e de dados ao mesmo tempo. Além disso, esse tipo de hierarquia faz com que ocorram mais erros na busca de informações do que uma cache tradicional, tendo em vista que mesmas posições de memória na cache serão alocadas para instruções e dados ao mesmo tempo.

O foco deste trabalho será uma hierarquia de memória cache de dois níveis, sendo o primeiro nível separado e o segundo nível unificado, como demonstrado na figura 2.6.

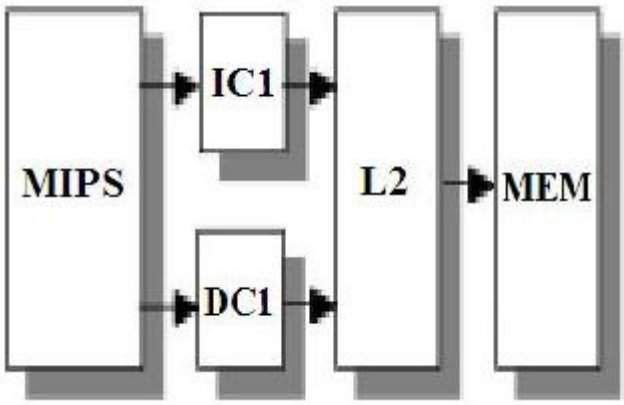

Figura 2. 7: Arquitetura de memória cache de dois níveis, com o segundo nível unificado.

Capítulo 3

Simuladores de Arquitetura

A fabricação de apenas 1 chipset possui um custo muito elevado, invibializando a confecção de unidades de diferentes chipsets que tenham a finalidade de realizar testes de desempenho. Devido a isso, surge o desenvolvimento de softwares que simulam arquiteturas variando desde sistemas operacionais, linhas de produção e sistemas de automação industrial, até arquiteturas de memória e de sistemas embarcados. Esses softwares ajudam no controle de gastos desnecessários, minimizando erros de projeto e aumentando a economia de energia, fazendo com que se possa diminuir a área necessária para sistemas de diversas aplicações, como o palmtop, por exemplo.

Neste capítulo serão citados alguns dos simuladores de arquiteturas mais utilizados, com uma breve explicação sobre cada um deles, dando ênfase ao que será utilizado para o alcance dos objetivos deste trabalho.

3.1 – Simuladores

Dentre os diversos simuladores existentes no mercado, podem-se destacar alguns por gerarem informações de consumo de energia e desempenho de memórias cache, assim como a área ocupada pelo circuito final da cache. Podemos citar:

- Platune: é uma ADL (Linguagem de Descrição de Arquitetura). Ferramenta utilizada para obtenção de parâmetros de potência e desempenho em uma plataforma SoC(*System on Chip*), geralmente sistemas embarcados, permitindo a análise através da entradas dos parâmetros configuráveis da plataforma

específica com o intuito de ter como resposta valores de desempenho e potência para a aplicação analisada[3].

- Cprof: é um sistema que identifica, nos códigos fontes de aplicações que são utilizadas para fazer testes completos na cache, as linhas e estruturas de dados que causam freqüentes erros de busca na cache, permitindo ao desenvolvedor, através da análise desses parâmetros, rever seu projeto e alterar algoritmos, afim de diminuir o número desses erros. O sistema Cprof consiste de dois programas: o simulador de cache uniprocessador Cprof, e o Xcprof, uma interface com o usuário XWindows. Esta ferramenta faz parte da WARTS (*Wisconsin Architectural Research Tool Set*), que é um conjunto de ferramentas utilizadas para simulação de arquiteturas de computadores[3].
- Tycho: avalia muitas alternativas de caches com um único processador, mas impõe algumas restrições severas às opções do projeto. Produz uma tabela de relações de faltas para caches de muitos tamanhos e associatividades, provendo que todas as caches tenham o mesmo tamanho de bloco (linha), não faz prefetching (pré-busca), e usam substituição LRU[3].
- DineroIII: é um simulador que, ao contrário do Tycho, avalia somente uma cache uniprocessador por vez, mas produz mais métricas de desempenho e permite mais opções de variação no projeto da cache [3].
- Expression: é uma ADL direcionada em exploração do espaço de projeto no nível de arquitetura para SoCs. A disponibilidade de uma variedade de núcleos de processadores de classes de arquiteturas diferentes fornece para o projetista do sistema um grande espaço de exploração para escolha mais adequada da base da arquitetura do processador que deseja-se projetar. Simula sobre a arquitetura desejada e finalmente gera informações sobre área, potência e desempenho incluindo quantidade de ciclos e estatísticas de uso da memória [3].

Nas seções 3.2 e 3.3 serão analisados o simulador SimpleScalar de memórias cache e a ferramenta eCACTI utilizados neste trabalho devido a obtenção dos resultados necessário para a análise do seu objetivo principal. O SimpleScalar, permitindo a entrada de parâmetros para configuração de caches de 2 níveis, tanto unificadas quanto separadas, de forma simples, e sendo muito utilizado no meio cientifico. Seus resultados

expressam confiança para os diversos tipos de configuração, expondo parâmetros de saída dos mais variados tipos de informação.

O eCACTI fornece as informações de área ocupada pelos circuitos da configuração de entrada, assim como os parâmetros de ciclos de máquinas necessários para a realização dos testes, além de ser um dos poucos, se não o único, a fornecer o valor separado do consumo de energia das componentes estática e dinâmica dos circuitos de memória cache. A componente dinâmica, no modelo de um transistor, equivale à carga e descarga de um capacitor, resultante da mudança de um bit de zero para um ou de um para zero. Já a componente estática de energia ocorre independente do chaveamento dos bits. A distinção entre estas componentes nos projetos que envolvem as mais recentes tecnologias de transistores (na ordem de nanometros) se torna importante, tendo em vista que à poucos anos atrás a componente estática era desprezível, no entanto, este panorama tem modificado (devido ao efeito capacitivo e aumento substancial da freqüência) fazendo com que a componente estática fosse substancialmente considerada nos projetos de sistemas embarcados.

3.2 – SimpleScalar

O SimpleScalar é um conjunto de ferramentas públicas, que suportam simulações detalhadas para algumas características encontradas nos processadores modernos. Possui 8 simuladores, desde um simulador funcional simples e rápido até um simulador de processadores superescalares. São eles: sim-fast, sim-safe, sim-profile, sim-cache, sim-cheetah, sim-eio, sim-bpred e o sim-outorder. Dentre esses simuladores, existem 2 deles, o sim-cheetah e o sim-cache, que são específicos para análise de memórias cache, gerando parâmetros de erros e acertos. As vantagens das ferramentas do SimpleScalar são sua alta flexibilidade, portabilidade, extensibilidade e desempenho[3].

O sim-cheetah gera resultados de simulação para múltiplas configurações de cache em uma única execução da simulação, abordagem esta chamada de *Single-Pass Simulation*, ao contrário de outros simuladores, sendo necessário uma grande quantidade de memória para a simulação de aplicações mais complexas. Isto se torna eficaz para um fluxo de instrução somente de leitura, porém não sendo indicado para

caches *write-back*, por se tornar mais custosa a substituição de um bloco ao ser escrito na volta para a memória RAM.

O sim-cache recebe parâmetros de entrada das configurações da memória cache, indicando quantidade de níveis, e se esses níveis são unificados ou separados. Ainda recebe a aplicação que será usada na execução da simulação para a medição das informações relacionadas a aspectos diversos dos acertos e erros ocorridos no decorrer da simulação.

O sim-cache recebe o seguinte formato de configuração <config> de memórias:

<name>:<nsets>:<bsize>:<assoc>:<repl>

Onde:

- <name> : nome da cache, deve ser único
- <nsets> : número de conjuntos na cache
- <bsize> : tamanho da linha
- <assoc> : associatividade da cache (potência de dois)
- <repl> : política de substituição (l,f,r), sendo: l=LRU, f=FIFO, r= randômico/aleatório

Como a arquitetura a ser simulada é de dois níveis, com o segundo nível unificado, utilizam-se as seguintes denominações:

- -cache:IL1 <config> => configura uma cache de instrução de nível um.
- -cache:DL1 <config> => configura uma cache de dados de nível um.
- -cache:IL2 DL2 –cache: DL2 <config> => configura uma cache unificada de nível dois.

É necessário indicar um arquivo onde o resultado final deverá ser armazenado, arquivo esse que será indicado pela inserção na linha de comando da tag “-redir:sim [caminho com o nome do arquivo]”, que deverá ser analisado e utilizado para o cálculo das demais informações, pois é nesse arquivo que se encontrarão a informações estatísticas de cada componente oriundas da simulação realizada.

Na figura 3.1 é mostrado um exemplo de uma linha de comando do SimpleScalar utilizando o simulador sim-cache, que será o simulador usado no decorrer deste trabalho por gerar as mais diversas informações das configurações simuladas dos

números de erros e acertos para cada uma delas, sendo necessário, a execução de um comando para cada nova configuração de memória cache.

Neste exemplo, é simulada uma cache de 2 níveis com o segundo nível unificado, contendo 128 conjuntos de cache, com tamanho de linha de 16 bytes, associatividade 1 e política de substituição do tipo aleatório, tanto para a cache de nível 1 de instruções IL1 quanto para a cache de nível 1 de dados DL1, e para a cache unificada UL2 de segundo nível, temos 1024 conjuntos de cache, com 16 bytes de tamanho de linha e associatividade 1, com política de substituição do tipo aleatório também. Note que, os resultados, da simulação para apenas esta configuração de cache, estão sendo armazenados em um arquivo chamado "conf1.txt" e que esta sendo utilizada a aplicação Bitcount, que será explicada no capitulo 4, que recebe o parâmetro 75000 para sua execução.

```
./sim-cache -redir:sim ./conf1.txt -cache:il1 il1:128:16:1:r
  -cache:dl1 dl1:128:16:1:r -cache:il2 dl2 -cache:dl2 ul2:1024:16:1:r
  ./bitcnts 75000
```

Figura 3.1: Exemplo de comando no SimpleScalar

Se faz necessário também a inclusão na linha de comando de um programa que gerará os dados para a simulação. Esse aplicativo deve ser acrescentado no fim da linha de comando seguido dos parâmetros necessários para a sua execução, como mostrado também na figura 3.1. Podemos visualizar no anexo 1, um exemplo de arquivo contendo as saídas de SimpleScalar.

É através das métricas de saída do sim-cache que tem-se a quantidade de acessos que ocorreram na cache L1 e na L2, assim como a quantidade de erros e acertos ocorridos, tanto na cache L1, onde se torna necessário o acesso a L2, quanto na cache L2, sendo necessário o acesso a memória principal do sistema. E, ao juntar essas informações com as informações oriundas do eCACTI, aplicando-as a um conjunto de fórmulas matemáticas, tem-se os valores necessários para a obtenção das conclusões deste trabalho.

3.3 – eCACTI

O eCACTI é uma melhoria do modelo CACTI, que é um modelo analítico de memória cache que gera resultados da energia consumida por acesso, tempo de acesso e área de um componente de memória cache, corrigindo algumas limitações observadas que levavam a um cálculo que desprezava a componente estática e fixava o valor das cargas capacitivas, gerando uma imprecisão no modelo de potência, principalmente quando se trata de nano-tecnologia(DSM). O eCACTI é utilizado para o cálculo, através de parâmetros de entrada, da potência consumida pela memória, tanto dinâmica quando estática, de cada modulo de memória cache, e da área ocupada, assim como informações de consumo de tempo no decorrer das operações realizadas na cache[3].

Para executar o eCACTI, são necessários os seguintes parâmetros:

<C><B><A><TECH><NSubbanks>

Onde:

- <C> : Tamanho da cache em bytes
- <B> : Tamanho da linha da cache em bytes
- <A> : Associatividade da Cache
- <TECH> : Tecnologia de transistor dada em micrometros
- <NSubbanks> : Número de Subbancos (matriz de armazenamento)

Pode-se ainda acrescentar ao comando do eCACTI uma configuração de organização da cache, além da configuração de parâmetros de entrada descrita acima, a partir de um arquivo que conterá estas especificações, ou executar o comando sem acrescentá-la, onde, nesta segunda opção, o próprio eCACTI avaliará a melhor configuração de organização para uma otimização levando-se em conta os parâmetros de avaliação, que são a área, o tempo e a potência consumidas[3]. Gerando, com isso, os demais resultados da simulação baseados nesta configuração otimizada. Parâmetros estes que se encontram no formato:

(Ndwl, Nspd, Ndbl, Ntwl, Ntspd, Ntbl)

Onde:

Ndwl : tamanho do segmento de *wordline*(array de dados);

Nspd : tamanho do segmento de *bitline*(array de dados);

Ndbl : número de conjuntos mapeados para casa *wordline*(array de dados);

Ntwl : tamanho do segmento de *wordline*(array de instruções);

Ntspd : tamanho do segmento de *bitline*(array de instruções);

Ntbl : número de conjuntos mapeados para casa *wordline*(array de instruções).

A figura 3.2 mostra um exemplo de linha de comando utilizando uma configuração especifica para a configuração da cache.

```
./eCACTI 2048 16 1 0.180000 1 (1,1,2,4,1,1)
```

Figura 3. 2: exemplo de eCacti com parâmetros de configuração.

Tem-se a possibilidade de inserir argumentos específicos, junta ou separadamente ao anterior, no mesmo arquivo de parâmetros, para o tipo de avaliação e resultados que se deseja obter do eCACTI, além do especificado acima, que seria a opção "-verbose", onde através deste argumento, o eCACTI mostrará os resultados completos de sua simulação, com detalhes de dissipação de potência estática e dinâmica, consumo de tempos de acesso e áreas ocupadas pelo banco de memória, além de gerar configurações otimizadas para cada um de seus componentes de análise: área, potencia e tempo.

Esses resultados também podem ser armazenados em um arquivo, o qual deve ser adicionado, o seu caminho, ao final da linha de comando do eCACTI. Na figura 3.2, tem-se um exemplo de uma execução do eCACTI, onde executa a simulação pelo eCACTI para uma cache de 2048 bytes, com o tamanho de sua linha de 16 bytes, com associatividade 1, com uma tecnologia de 0,18 micrometros e apenas 1 subbanco, possuindo um arquivo de parâmetros. Contendo no anexo 2, um exemplo de arquivo com os resultados de eCACTI.

```
./eCACTI 2048 16 1 0.180000 1 paramFile
```

Figura 3.3: exemplo de eCACTI.

Capítulo 4

Metodologia Proposta

Neste capítulo será visto uma descrição da metodologia adotada para a realização deste trabalho, com a explicação do funcionamento de cada etapa relacionada.

A figura 4.1 demonstra uma visão geral do fluxo de projeto que é composto pelas seguintes etapas: 1 – Especificação de Benchmark; 2 – Geração de Configurações de cache; 3 – Simulação de Configurações; 4 – Geração de Dados do eCACTI; 5 – Desenvolvimento de Aplicação; 6 – Transferência de Dados para o Banco de Dados; 7 – Aplicação de Fórmulas; 8 – Execução da Heurística; 9 – Geração de Resultados. Sendo as etapas 3, 4 e 5 possíveis de serem realizadas em paralelo, como mostrado na figura 4.1. A etapa 9 será explicada no próximo capítulo.

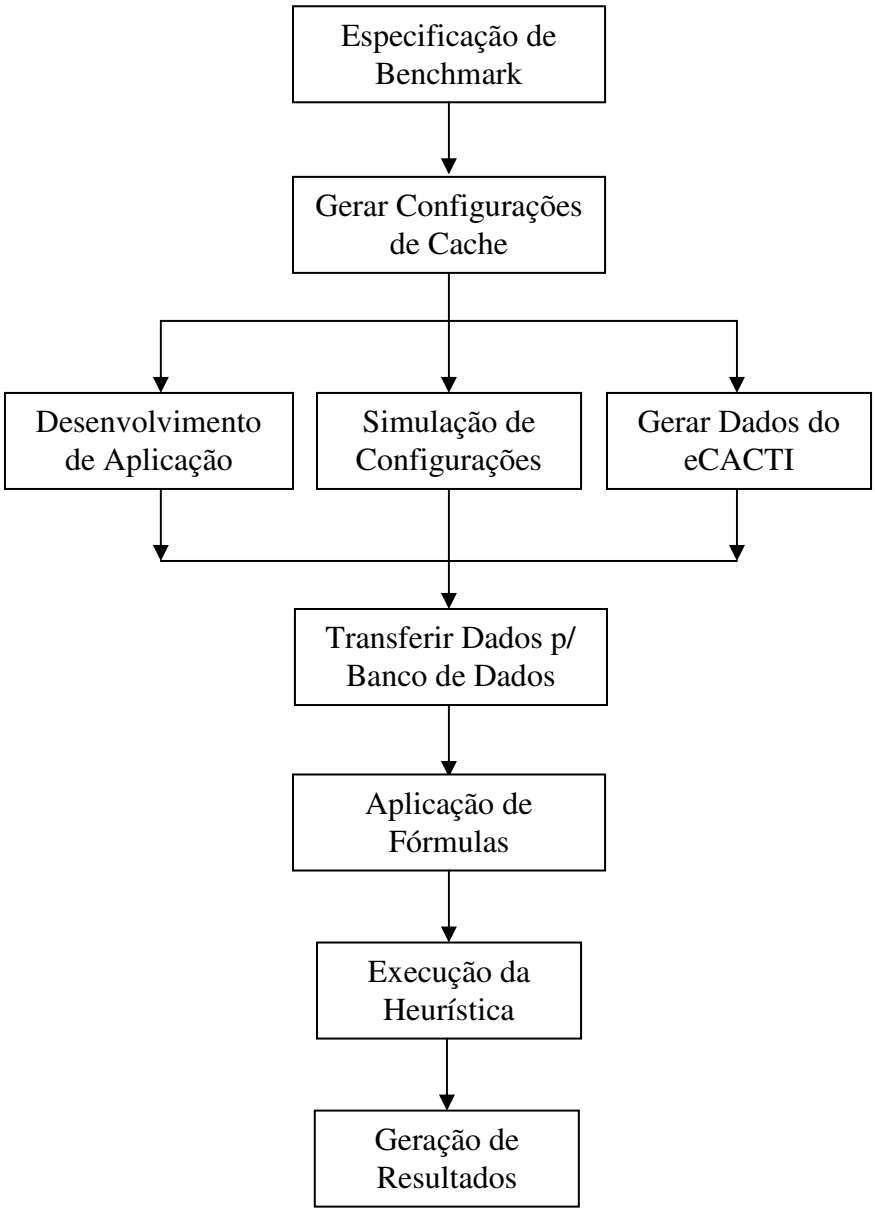

Figura 4.1: Visão geral do Fluxo de Projeto

4.1 – Especificação de Benchmark

Utilizando o *Benchmark Mibench Suite*, que é normalmente utilizado pelo meio científico para avaliar sistemas embarcados tradicionais e de varias outras categorias (tais como telecomunicações, automotiva, consumo, segurança e rede), para que seja possível, através dessas aplicações, validar o objetivo deste trabalho,

comprovando, ou não, a influência da área da memória cache no desempenho e consumo de energia do processador.

Foram utilizadas três aplicações do *Benchmark Mibench Suite*: o BitCount, o Susan e o Dijkstra. Essas aplicações estão mais bem explicadas na tabela 4.1, que foi retirada de [3] e adaptada para o nosso objetivo, retirando as aplicações que não foram utilizadas neste trabalho.

Tabela 4.1: Aplicações do Benchmark

Aplicação	Categoria	Descrição
Bitcount Small	Controle Automotivo e Industrial	Realiza testes de habilidade de manipulação de bits de um processador, contando o número de bits em um vetor de inteiro.
Susan Small	Controle Automotivo e Industrial	Faz parte de um pacote de reconhecimento de imagem. Foi desenvolvido para reconhecimento de bordas e curvas em uma Imagem de Ressonância Magnética de um cérebro.
Dijkstra Small	Network	Constrói um grande grafo em uma representação de matriz adjacência e calcula o menor de todos os paths entre todos os pares de nós usando aplicações repetidas do algoritmo Dijkstra.

4.2 – Gerar Configurações de Cache

Foi utilizado um programa desenvolvido na linguagem C, aplicação essa utilizada em [3] e adaptada para este trabalho, pois em [3] a análise feita foi baseada em uma arquitetura de dois níveis de memória cache, com ambos os níveis separados, e a arquitetura analisada neste trabalho tem seu segundo nível unificado. Ao ser executado, o programa gera um *script* contendo as linhas de comandos de execução, para cada uma das aplicações do *Benchmark*, com as possíveis configurações de memória cache. Configurações determinadas pela variação de três parâmetros, que são utilizados no SimpleScalar: o número de conjuntos da cache, o tamanho da linha da cache e a

associatividade da mesma. Sendo que, o número de conjuntos existentes em uma configuração de cache, é dado pela equação (1), que envolve no cálculo, também, o tamanho da cache.

Núm. Conjuntos Cache = Tamanho Cache / (Tamanho Linha * Associatividade)	(1)

Esta aplicação em C também verifica se a configuração gerada é válida no âmbito dos conceitos de utilização de memórias cache, pois, para uma configuração de memória cache ser válida, ela precisa obedecer aos seguintes parâmetros:

- O Tamanho da cache L2 deve ser maior ou igual ao Tamanho da cache L1 de Dados e ao tamanho da cache L1 de Instruções;
- O Tamanho da Linha de L2 deve ser maior do que o Tamanho da Linha da cache L1 de Dados e do que o Tamanho da Linha da cache L1 de Instruções;
- O Número de Conjuntos de L2 deve ser maior do que o Número de Conjuntos da cache L1 de Dados e do que o Número de Conjuntos da cache L1 de Instruções;

As configurações dos parâmetros da cache foram limitadas em valores mínimos e máximos nos quais os tamanhos das caches de primeiro nível variam entre 2 e 8 kbytes, e o tamanho da cache de segundo nível varia entre 16 e 64 kbytes. Já o tamanho das linhas para ambos os níveis de cache foi variado entre 16 e 64 bytes e a associatividade entre 1 e 4. Todos os parâmetros sempre variando exponencialmente, com potência de 2. Por exemplo, o tamanho da associatividade existente nas configurações é sempre 1, 2 ou 4, assim como o tamanho da cache de primeiro nível, recebendo os valores de 2 kbytes, 4 kbytes e 8 kbytes.

Após a verificação das combinações possíveis e a validação delas pela aplicação, foram geradas 9084 combinações, conseqüentemente a mesma quantidade de linhas de comando a serem executadas pelo *script*. Para cada uma das três aplicações utilizadas do *Mibench Benchmark*, foi gerado um arquivo de *script*, com extensão ".sh", que, ao ser executado, gerará as 9084 saídas do simulador SimpleScalar para cada uma das aplicações.

Foi gerado também um arquivo de *script*, através de outro programa em C, com as configurações de cache para serem executadas através do eCACTI, configurações essas que também possuem seus limites máximos e mínimos para os três parâmetros utilizados pelo eCACTI, que são o tamanho da cache, o tamanho da linha e a associatividade. Os dois últimos limites variam em igual valor do *script* gerado para a execução no SimpleScalar, já o primeiro limite abrangeu o intervalo de 2 kbytes a 2 Mbytes, para que se possa associar, no devido momento, os resultados da execução desses *scripts* através da relação existente na equação (1), entre o número de conjuntos e o tamanho da cache, juntamente com a associatividade e o tamanho da linha. O script do eCACTI gera 99 combinações diferentes de memória cache, permanecendo sempre o mesmo valor no parâmetro de tecnologia utilizada e de número de subbancos de cache.

4.3 - Desenvolvimento de Aplicação

Foi desenvolvida uma aplicação em linguagem de programação Delphi®, tendo como propósito facilitar a transferência das informações nos arquivos de configuração gerados pelo SimpleScalar e pelo eCACTI para um banco de dados, e a reunião destas informações para melhor análise dos resultados e aplicação da heurística a ser utilizada.

Vale salientar que o objetivo deste trabalho não é o desenvolvimento de um software para executar a análise da heurística, devido a isto, não houve a preocupação com aspectos de interface, apenas com a funcionalidade da aplicação.

A figura 4.2 mostra a tela inicial da aplicação, onde é possível transferir as informações das simulações para um arquivo texto “.txt”, juntando todas elas em apenas um arquivo, e depois a transferência dessas informações para um banco de dados, minimizando assim o risco de ocorrência de falhas devido a manipulação de arquivos.

Figura 4.2: Tela Inicial do Programa de Migração de Dados

Na figura 4.3 é mostrada a tela onde é possivel visualizar os resultados de todas as configurações simuladas pelo Simplescalar, unificadas com as configurações simuladas do eCACTI, através da criação de uma *view*, que é uma seleção de informações específicas de tabelas variadas, unificando as mesmas pelas cláusulas condicionais pertinentes a estas junções, das tabelas do banco de dados.

A aplicação permite filtrar as informações, disponibilizando apenas os resultados referentes a uma das aplicações do *Benchmark*, e por meio dessa tela, podemos localizar as configurações desejadas aplicando-se a heurística, e, apertando o botão “inserir”, essas informações serão transferidas para uma nova lista de valores, onde serão comparados os parâmetros desejados.

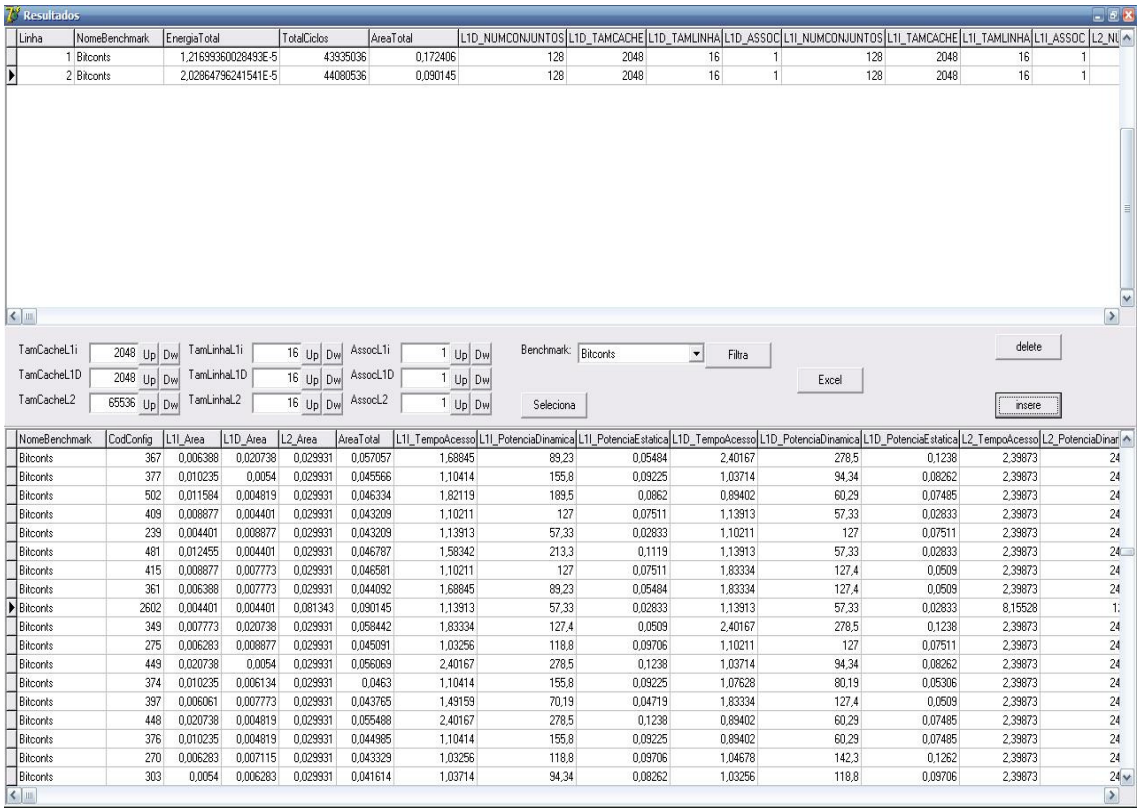

Figura 4.3: Tela de Visualização de Resultados das Simulações e de Seleção para a realização da heurística.

Também, na tela da figura 4.3, é possível transferir as informações para o Excel, para manipulá-las e gerar os gráficos de análise dos resultados, gráficos esses que também poderiam ser gerados pelo Delphi®, mas, mais uma vez, o objetivo principal deste trabalho não é esse, podendo ficar para um trabalho futuro o aprimoramento da aplicação.

4.4 - Simulação de Configurações

Após a geração dos scripts para cada uma das aplicações do Benchmark, executaremos os mesmos, um de cada vez, para que o SimpleScalar possa simular o funcionamento da cache de acordo com a aplicação e com a configuração de cache passadas como parâmetro.

Serão gerados 9084 arquivos de configuração, contendo as métricas de saídas da simulação do SimpleScalar para cada uma das aplicações selecionadas, que serão

armazenados em um local específico, com nomes padrões para identificar a configuração ao qual pertence e a partir de qual aplicação foi gerado.

Através do comando “sh” seguido do nome do arquivo de script, para que o nosso script seja executado. Por exemplo, digitando no prompt do terminal do Linux o comando “sh simula.sh”, executará todas as linhas de comando existentes em nosso script.

Para a simulação de todos os três scripts, despendeu-se um tempo relativamente grande, a saber: para a simulação da aplicação Bitcount foi necessário 24h para simular todas as configurações possíveis, para a aplicação Susan, foi necessário um tempo de 21h aproximadamente, e para o Dijkstra, um tempo de 36h para gerar os resultados das configurações existentes.

4.5 - Geração de Dados do eCACTI

Foi utilizado eCACTI, a fim de obter as métricas de energia dinâmica e estática, o tempo de acesso para cada instrução e o tamanho da área ocupada pela configuração da cache, dentre outras informações, que são imprescindíveis para a geração dos resultados deste trabalho.

Da mesma forma que ocorreu no simulador SimpleScalar, o *script* gerará para cada linha de comando, um arquivo para armazenar os parâmetros de saída do eCACTI, que devem ser armazenados em local especifico também, para que possam ser utilizados na aplicação desenvolvida.

O tempo despendido com relação a execução do script de linhas de comando para o eCACTI foi muito menor do que o gasto para a simulação das aplicações pelo SimpleScalar, sendo necessária menos de 1h de execução.

4.6 - Transferência de Dados para o Banco de Dados

Para facilitar a análise e manipulação desses resultados, foi criado um banco de dados para armazenamento das informações em tabelas, separadas por cada aplicação

do benchmark e pela configuração de cache de cada uma delas. A figura 4.4 mostra o diagrama ER (Entidade-Relacionamento) do banco de dados criado.

Figura 4.4: Diagrama ER do Banco de Dados

Foi utilizado o Banco de Dados Firebird para o armazenamento dessas informações, e, para a sua manipulação, o Sistema Gerenciador de Banco de Dados(SGBD) IBExpert e a aplicação desenvolvida.

Através da tela inicial, mostrada na figura 4.2, transfere-se as informações geradas pelo SimpleScalar, para cada uma das aplicações do *Benchmark*, armazenadas em arquivo, para a base de dados.

As informações referentes ao eCACTI serão inseridas na tabela chamada de eCACTI, os nomes dos *Benchmarks* serão inseridos na tabela de *Benchmarks*, e os dados oriundos das simulações do SimpleScalar, serão inseridos na tabela de resultados, onde cada um dos parâmetros de saída serão armazenados nos respectivos campos das tabelas.

A tabela eCACTI também possui um campo chamado de NumConjuntos, que foi criado para facilitar a junção dela com a de resultados do SimpleScalar. Esse campo é calculado para cada entrada de registro da tabela eCACTI.

4.7 – Aplicação de Fórmulas

Nas informações armazenadas no banco de dados, que são o resultado das simulações executadas, não estão contidas as informações de energia total, ciclos totais e área total, sendo necessária a aplicação de fórmulas para a geração desses valores.

O cálculo da área total ocupada pela memória cache para cada configuração simulada pelo SimpleScalar é a soma das áreas independentes das configurações de cada cache que compõe a arquitetura trabalhada, que neste caso é a área da cache de dados do primeiro nível, somando com a área da cache de instruções do primeiro nível e com a área da cache unificada do segundo nível. Cálculo este que se encontra na equação (2), dado em cm^2.

$$\textbf{Área Total } (cm^2) = \textbf{Área da cache LD1 } (cm^2) + \textbf{Área da cache LI1 } (cm^2) + \textbf{Área da cache L2 } (cm^2) \quad (2)$$

O cálculo da energia total consumida está exposto na equação (3). Essa componente se dá pela soma das energias consumidas por cada uma das caches, tanto a componente dinâmica quanto a estática. A esta soma será adicionado a energia gasta

para o acesso a memória principal do sistema. Lembrando que as energias são dadas em unidades de Joules(J) ou nanoJoules(nJ), as potências são dadas em miliwatts(mW) e o tempo em nanosegundos(ns).

Energia Total (J) = Energia Total Memória Principal (J) + Energia Total LD1 (J) + Energia Total LI1 (J) + Energia Total L2 (J)	(3)

Desmembrando a equação (3), tem-se o cálculo da energia total da memória principal, dada em Joules (J), na equação (4), que se dá pela multiplicação da energia gasta por cada acesso realizado na memória principal pelo número de palavras. A divisão por 10^9 é para conversão da unidade de (nJ) para (J).

Energia Total Memória Principal (J) = (Energia Memória Principal por Acesso (nJ) * Número de Palavras Lidas na Memória Principal)/10^9	(4)

Aprofundando um pouco mais nas componentes desta última fórmula, tem-se que a energia por acesso à memória principal é dada pela equação (5), onde a energia Dinâmica é somada com a energia Estática por acesso a LD1.

Energia Memória Principal por Acesso (nJ) = 3,1*(Energia Dinâmica de LD1 por Acesso (nJ) + Energia Estática de LD1 por Acesso (nJ))	(5)

Sendo o cálculo da Energia Dinâmica consumida por acesso dada pela equação (6), e o cálculo da Energia Estática consumidas por acesso dada pela equação (7), energias essas calculadas para cada componente da memória cache, tanto LD1, quanto LI1 e L2.

Energia Dinâmica / Acesso (nJ) = Potência Dinâmica (mW) * Tempo Acesso (ns)	(6)

Energia Estática por Acesso (nJ) = Potência Estática (mW) * Tempo de Acesso (ns)	(7)

Onde as componentes de Potência Dinâmica, Potência Estática e Tempo de Acesso são oriundos das simulações do eCACTI para cada uma das configurações de cache existentes.

Já o Número de Palavras Lidas na Memória Principal é dado pelo cálculo mostrado na equação (8) abaixo.

Número de Palavras Lidas na Memória Principal = Número de Faltas em L2 * Tamanho da Linha de L2 (8)

Onde, o Número de Faltas em L2 é gerado pela simulação das configurações no SimpleScalar para as configurações de entrada, e significa que o processador não encontrou a informação que queria na cache L2 e o controlador terá que acessar a memória principal para obter a informação. Já a componente de Tamanho da Linha de L2 é oriunda da própria configuração da cache L2, que serviu de entrada para a simulação no SimpleScalar e no eCACTI.

Retornando à equação (3), serão analisadas as componentes de Energia Total da cache LD1, LI1 e L2, pois esses cálculos têm o mesmo principio, apenas alterando a busca das informações inerentes a cada componente de memória. Nessa fórmula, para fins de generalização, a Energia Total L refere-se a essa componente, abrangendo qualquer nível e componente da cache. O cálculo dessa energia é mostrado na equação (9).

Energia Total L (J) = (Energia Dinâmica por Acesso (nJ) + Energia Estática por Acesso (nJ) + Energia Stall(nJ))/10^9 (9)

Onde essas energias já foram explicadas anteriormente, e a Energia Stall é uma constante que vale “0,6946752”, que equivale a energia devido às paradas no processador (stalls), tomamos como base no trabalho cedido por Ann-Gordon Ross para [3], para extração desse valor constante. O valor de “10^9” também já foi explicado anteriormente, que serve para transformar de nanoJoules para Joules.

O próximo passo é calcular a última componente de estudos deste trabalho, o Desempenho, definida através dos Ciclos Totais, pois quanto maior o número de ciclos necessários para executar uma aplicação, menor é o desempenho da configuração analisada. Esse cálculo se dá realizando-se a operação contida na equação (10).

Total Ciclos = Número Acesso L2 + Custo Falta L2 + Custo Falta L1 + ValorMáximo(Número Acesso LI1, Número Acesso LD1) (10)

O Número de Acessos a L2 da equação (10) é um dos resultados obtidos através da simulação do SimpleScalar para cada uma das suas configurações, indicando, como o próprio nome diz, a quantidade de vezes que a cache L2 foi acessada devido a falta ocorrida em L1.

O Custo da Falta em L2 significa a quantidade ciclos necessários para se buscar uma informação na memória principal quando essa informação não é encontrada na cache L2. É calculado pela equação (11), onde a componente chamada de Penalidade é uma constante de valor "44", que significa que são necessários 44 ciclos para que uma informação que não é encontrada em L2 seja trazida para o processador da memória principal. Esta constante é multiplicada pelo número de faltas que ocorreram em L2, que é trazido da simulação do SimpleScalar para as configurações especificadas.

Custo da Falta L2 = Penalidade * Número de Faltas em L2 (11)

O Custo de Falta em L1 significa o custo em ciclos de uma falta na cache L1, e é dado pela equação (12), indicando que a informação pode ser encontrada na cache L2 ou na memória principal.

Custo Falta L1 = (Número de Acertos em L2 * Ciclos por Acerto de L2) + (Taxa da Falta L2 (%) * Custo Falta L2) (12)

Onde o Número de Acertos em L2 é oriundo, mais uma vez, dos resultados da simulação no SimpleScalar para cada configuração de cache.

A componente "Ciclos por Acerto de L2" é uma constante de valor igual a "5" que indica a quantidade de ciclos decorridos quando se tem um acerto ocorrido na cache L2. Pode-se verificar a grande diferença dessa constante para a constante de penalidade, utilizada em (11).

A Taxa da Falta de L2 indica o percentual do número de faltas ocorridas em L2 pelo número total de acessos à L2, onde ambos os parâmetros são trazidos da simulação do SimpleScalar. Seu cálculo é mostrado na equação (13) abaixo.

Taxa de Falta L2 (%) = Número de Faltas de L2 / Número de Acessos a L2	(13)

O componente da equação (10) “ValorMáximo(Número Acesso LI1, Número Acesso LD1)”, é oriundo da arquitetura de cache analisada, uma cache de 2 níveis com o segundo nível unificado, mas com o primeiro nível separado. E como foi explicado no capítulo 3, esta arquitetura tem suas vantagens e desvantagens, sendo o ValorMaximo calculado para esta equação, uma das desvantagens, pois, contrariamente ao que ocorre na cache de dois níveis separados, onde o acesso é paralelo, aqui o acesso a cache L2 é seqüencial, não nos permitindo fazer o cálculo separado das contribuições dos acessos a LD1 e a LI1. Sendo assim, eles devem ser verificados ao mesmo tempo, utilizando o maior valor entre os dois, pois, é o que trará maior custo para a busca na cache. Os parâmetros para esta função são trazidos da simulação do SimpleScalar.

4.8 – Execução da Heurística

Não sendo a heurística o foco principal deste trabalho, assim como sua elaboração, foi utilizado uma heurística pré-existente, a TECH-CYCLES[3], desenvolvida para cache de dois níveis, com o segundo nível separado.

A heurística TECH-CYCLES será utilizada duas vezes para cada um dos aplicativos do *Benchmark*: i) analisando apenas os componentes ciclos totais e energia total, em conjunto; ii) analisando a componente área total, em conjunto com as anteriores, para verificação de sua influência na escolha da melhor configuração de cache.

A figura 4.5 demonstra, a partir de um fluxograma, o funcionamento dessa heurística, sabendo que: i) o tamanho da cache irá variar de 64 até 16 kbytes, para a L2, e de 2 até 8 kbytes para L1D e L1I; ii) o tamanho da linha irá variar de 16 até 64 bytes; iii) a associatividade irá variar de 1 até 4. As variações são em potência de 2.

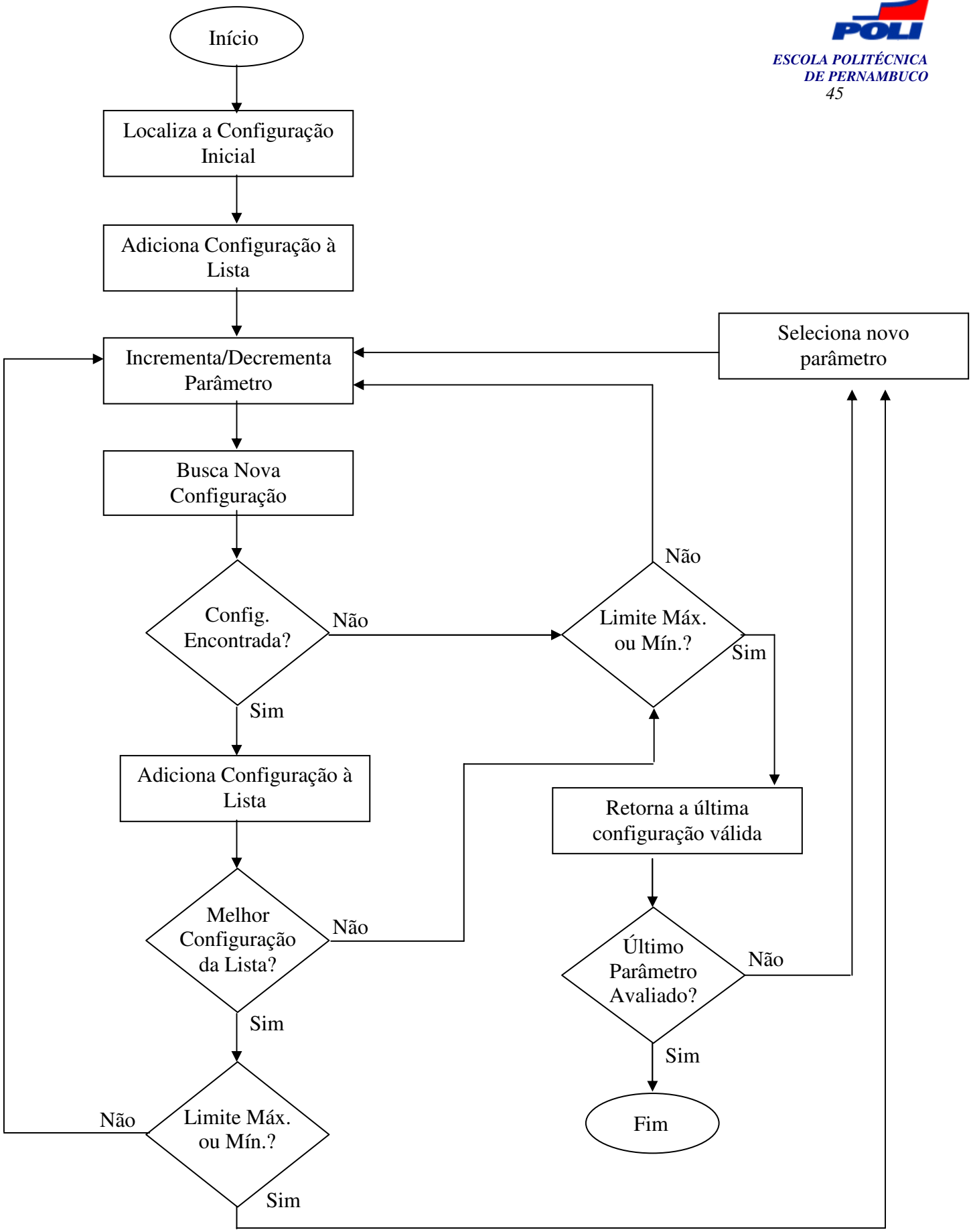

Figura 4.5: Fluxograma de Funcionamento da Heurística TECH-CYCLES

O primeiro passo desta heurística é a configuração dos valores iniciais de cada parâmetro, onde L1D e L1I receberão seus valores de tamanho mínimos, 2 kbytes, e a L2 receberá o seu valor máximo, 64 kbytes. O tamanho de linha e a associatividade, de todas elas, iniciarão com os valores mínimos também, 16 bytes para a linha e 1 para a associatividade.

Então, busca-se esta configuração dentre todas da aplicação selecionada, o Bitcount, o Susan ou o Dijkstra, e, se a mesma existir, será inserida em uma nova lista, lista de configurações selecionadas pela heurística, servindo como base para comparações com as próximas seleções.

O próximo passo é diminuir o tamanho da cache L2, de 64 para 32 kbytes, localiza-se a nova configuração. Se for encontrada, será adiciona a lista de configurações da heurística e, comparam-se os componentes da análise, sendo, nesta primeira etapa, a energia total e os ciclos totais. Se a nova configuração possuir ambos os valores menores do que a configuração base, esta será tomada, a partir de agora, como base para as novas comparações, seguindo para o próximo passo, continuando a diminuir o tamanho da L2 para 16 kbytes, para que possa ser realizada uma nova comparação de valores. Caso contrário, retorna-se a configuração anterior e começa-se a alterar o próximo parâmetro, que será o tamanho da linha da cache.

Um outro caso em que começa-se a alterar o próximo parâmetro é quando chega-se ao limite máximo ou mínimo de uma das variações de valores.

Após alterar o tamanho da cache L2 e encontrar a melhor configuração para ela, prossegue-se alterando o valor do tamanho da linha de L2, observando os limites de alteração, do mínimo ao máximo ou até encontrar uma configuração que tenha os componentes comparados piores do que a configuração base.

Logo depois, segue-se, com este mesmo principio, alterando a associatividade da cache L2. Encontrando a melhor configuração para a L2, começa-se a alterar a L1D, seguindo a mesma seqüência de alteração de valores dos parâmetros, e depois passa a alterar a L1I.

Quando o último parâmetro for alterado, a associatividade da L1I, chegasse ao fim da execução da heurística, obtendo uma lista de configurações, onde a melhor delas que será a última configuração base da análise, pois possui os menores valores de ciclos totais e de energia total.

Estas informações serão transferidas para uma planilha eletrônica, onde poderemos gerar os gráficos desejados para a melhor visualização dos resultados. Gráficos estes que serão mostrados no capítulo 5.

Após o termino da análise para os componentes de ciclos totais e de energia total, repete-se a heurística acrescentando na análise, além dessas duas componentes mencionadas, a área total ocupada pela configuração. Isso não altera em nada o funcionamento do fluxograma, tendo apenas que comparar essas 3 componentes em vez de 2, onde os valores dessas 3 componentes devem diminuir, na comparação com a configuração base, para que seja considerada a melhor configuração.

Todos esses passos são realizados, de forma mais simples, através da aplicação desenvolvida para este fim, na tela da figura 4.3, onde se deve fazer a filtragem necessária para cada uma das aplicações do *Benchmark*, pois a análise realizada pela heurística deve ser feita para cada uma dessas aplicações, separadamente.

Capítulo 5

Estudo de Casos e Resultados

Neste capítulo será visto o ambiente experimental no qual foram executadas as aplicações para os estudos do objetivo deste trabalho, mostrando as configurações do computador utilizado e o sistema operacional no qual as aplicações foram executadas, além dos resultados obtidos da aplicação da heurística sobre os dados das simulações realizadas.

5.1 – Ambiente Experimental

Para a simulação das aplicações citadas no capitulo anterior, foi utilizado o Sistema Operacional Linux, na distribuição Ubuntu 8.0, pois tanto o SimpleScalar quanto o eCACTI só podem ser instalados em ambiente Linux. O motivo de utilização desta versão do Ubuntu, é pela mesma ter-se apresentado mais estável do que as anteriores.

O equipamento utilizado para a realização das simulações foi um *notebook* da marca CCE, com a seguinte configuração: Processador Intel Core 2 Duo T5300 de 1.73GHz, 1GByte de memória RAM DDR2 de 667MHz, HD SATA de 120GBytes, com placa mãe on-board utilizando 64MBytes para o processamento de vídeo.

5.2 - Resultados

Após a aplicação da Heurística TECH-CYCLES, foram geradas duas novas listas de configurações para cada uma das aplicações do *Benchmark*, uma que analisa apenas os ciclos totais com a energia total, e a outra que analisa as duas componentes anteriores juntamente com a área total.

Para melhor visualizar os efeitos da área nas análises a serem realizadas dos resultados encontrados, foi criada uma Função de Custo (FC) que é a multiplicação das componentes analisadas neste trabalho: o Desempenho, a Energia Total e a Área Total. Este cálculo esta mostrado na equação 14, abaixo.

Função de Custo (FC) = Ciclos Totais * Energia Total * Área Total (14)

Foi criada uma informação de Valor em Reais, do custo da implementação da configuração analisada. O valor da implementação de 1MByte de memória cache é de U$ 20,00(vinte dolares)[5], valor este que transformado para Reais, tomando como base o valor médio do dólar paralelo, que tem oscilado em torno de R$ 1,75(um real e setenta e cinco centavos). Transformando de dólares para reais, temos o valor de R$35,00(trinta e cinco reais) para cada MByte de memória cache construída. Como nossas configurações estão limitadas a unidade de Kbytes, temos que o valor do kbyte é de R$ 0,03418.

A equação (15) mostra como calcular o valor, em reais, para uma configuração de cache. O valor de 1024 presente em (15) é devido à conversão da unidade do tamanho da cache de Bytes para Kbytes.

Valor (R$) = ((Tamanho Cache LD1/1024) + (Tamanho Cache LI1/1024) + (Tamanho Cache L2/1024)) * ValorPorKByteCache (15)

Vale salientar também que todos os gráficos gerados a partir das análises na aplicação da heurística e configurações de cache, são em função da quantidade de ciclos totais, encontrados no eixo x, e da energia total, encontrada no eixo y dos gráficos. Estes gráficos possuirão um padrão para a representação dos pontos das configurações de

cache, onde os pontos quadrados são as informações selecionadas pela heurística para a análise sem a área total, os pontos triangulares são as informações da análise com a área total e os pontos em forma de losângulo representam todas as configurações de cache obtidas das aplicações do *Benchmark* a ser analisada.

Em cada uma das tabelas apresentadas existe uma linha em destaque, indicando que esta é a melhor configuração encontrada pela execução da heurística, como demonstrado na seção 4.8 do capítulo 4. Essa melhor configuração tem como critério o menor número de ciclos totais e a menor energia total, em conjunto, quando a heurística é realizada sem a influência da área. Já nos casos em que a heurística é executada com análise da influência da área, o critério é o menor valor, em conjunto, do número de ciclos totais, da energia total e da área.

Estão mostrados, na tabela 5.1, as configurações adquiridas com a heurística utilizada com os resultados da aplicação Bitcount, sem levar em consideração a análise da área para a execução da heurística e da busca pela melhor configuração. Logo em seguida temos a tabela 5.2 que mostra as configurações da análise executando a heurística com a influência da área total.

Tabela 5.1: Resultado da Heurística sem análise da influência da área em sua avaliação, utilizando os resultados do Bitcount.

Config. Cache LI1	Config. Cache LD1	Config. Cache L2	Ciclos Totais	Energia Total (µJ)	Área Total (cm^2)	FC	Valor (R$)
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	43935036	12,1699360028493	0,172406	92,1832	2,324
(2048, 16, 1)	(2048, 16, 1)	(32768, 16, 1)	44080536	20,2864796241541	0,090145	80,6112	1,230
(2048, 16, 1)	(2048, 16, 1)	(65536, 32, 1)	43856495	14,0489618526754	0,121079	74,6014	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	43918997	11,1363592781345	0,121503	59,4268	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	43926316	11,5396997390787	0,097749	49,5486	2,324
(2048, 16, 1)	(4096, 16, 1)	(65536, 16, 2)	43903342	40,0709416168946	0,124875	219,6861	2,393
(2048, 16, 1)	(2048, 16, 2)	(65536, 16, 2)	43912650	14,8472267508868	0,123236	80,3475	2,324
(4096, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	43912533	10,9712470918450	0,124875	60,1617	2,393
(8192, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	43911511	11,0689064219797	0,137840	66,9975	2,529
(4096, 16, 2)	(2048, 16, 1)	(65536, 16, 2)	43915077	11,1526486639525	0,123490	60,4816	2,393

Tabela 5.2: Resultado da Heurística com análise da influência da área em sua avaliação, utilizando os resultados do Bitcount.

Config. Cache LI1	Config. Cache LD1	Config. Cache L2	Ciclos Totais	Energia Total (µJ)	Área Total (cm^2)	FC	Valor (R$)
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	43935036	12,1699360028493	0,172406	92,1832	2,324
(2048, 16, 1)	(2048, 16, 1)	(32768, 16, 1)	44080536	20,2864796241541	0,090145	80,6112	1,230
(2048, 16, 1)	(2048, 16, 1)	(65536, 32, 1)	43856495	14,0489618526754	0,121079	74,6014	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	43918997	11,1363592781345	0,121503	59,4268	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	43926316	11,5396997390787	0,097749	49,5486	2,324
(2048, 16, 1)	(4096, 16, 1)	(65536, 16, 2)	43903342	40,0709416168946	0,124875	219,6861	2,393
(2048, 16, 1)	(2048, 16, 2)	(65536, 16, 2)	43912650	14,8472267508868	0,123236	80,3475	2,324
(4096, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	43912533	10,9712470918450	0,124875	60,1617	2,393
(2048, 16, 2)	(2048, 16, 1)	(65536, 16, 2)	43915236	11,0553604464191	0,123236	59,8309	2,324

Ao comparar as tabelas 5.1 e 5.2, que representam os resultados da heurística, observa-se que as configurações encontradas são as mesmas, uma vez que a heurística é a mesma, mas, na execução dela com a análise de área em conjunto com a energia e os ciclos, tem-se uma redução no número de linhas da tabela 5.2, indicando que a heurística convergiu mais rápido para um resultado final.

Apesar disso, pode-se verificar que a função de custo não é a menor dentre todas as configurações das tabelas, mas o valor em reais relacionado à construção da área da configuração da memória cache analisada, é menor no caso da tabela 5.2. Sendo as configurações em destaque, nas tabelas 5.1 e 5.2, diferentes, observa-se que a inclusão da área na análise da execução da heurística influencia na seleção da melhor configuração.

Na figura 5.1 temos os gráficos gerados pelos pontos dos resultados da heurística utilizada, para melhor visualização, referindo-se as tabelas 5.1 e 5.2..

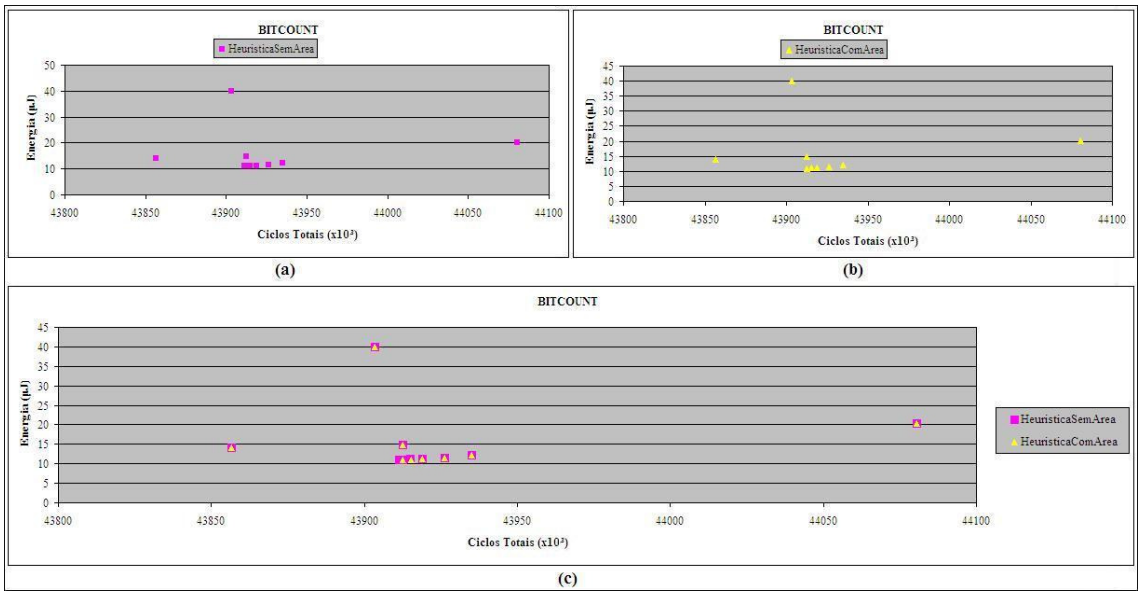

Figura 5.1: Gráficos das configurações selecionadas pela heurística TECH-CYCLES, referentes às tabelas 5.1 e 5.2, onde: (a) Aplicação da heurística sem análise de Área Total; (b) Aplicação da heurística com análise de Área Total; (c) Junção das duas análises.

O gráfico exibido na figura 5.2 mostra a junção de todos os pontos gerados pelas configurações obtidas do Bitcount e as configurações vindas da aplicação da heurística

mostradas na figura 5.1, facilitando a visualização da localização dos pontos obtidos pela heurística com relação ao espaço amostral completo.

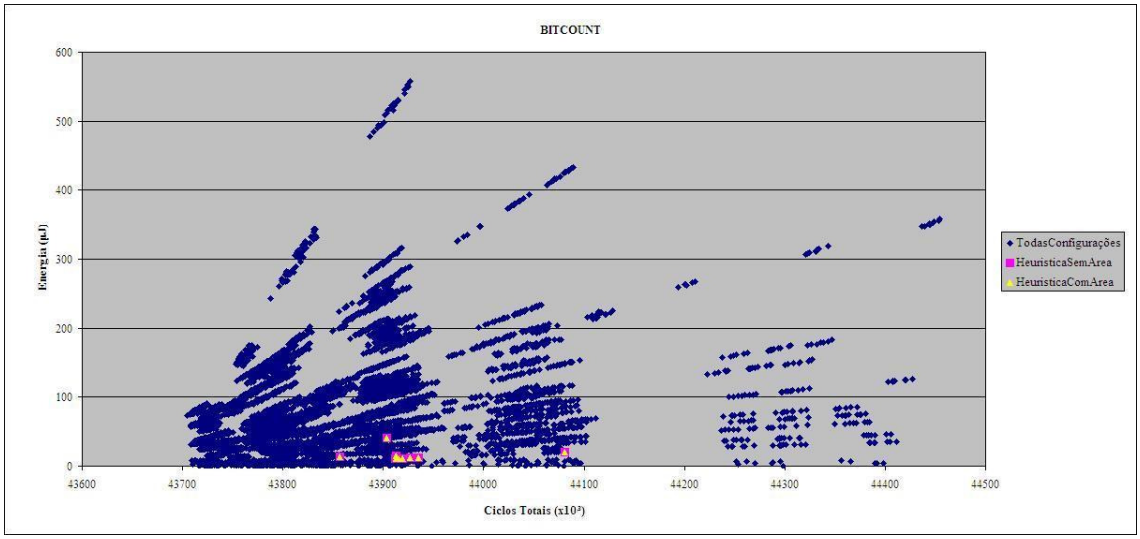

Figura 5.2: Representação gráfica das configurações obtidas do Bitcount em confrontamento com as obtidas pela heurística com análise da área e sem a análise da área.

Observar-se, a primeira vista, através da figura 5.2, que a heurística aplicada não gera os melhores resultados, ou, pelo menos, os resultados esperados, próximos dos pontos ótimos, que são a menor energia e o menor número de ciclos juntamente com a menor área, reforçando a necessidade de uma heurística própria para esta configuração e para a análise da área em conjunto com a energia e com os ciclos totais.

Fazendo a mesma análise para os resultados obtidos da simulação do SimpleScalar e do eCACTI para o Susan, tem-se, nas tabelas 5.3 e 5.4, os dados referentes as configurações selecionadas pela heurística TECH-CYCLES.

Pode-se perceber que, diferentemente do que ocorreu no Bitcount, a aplicação da heurística, com a análise e sem a análise da área, não modificou a escolha da melhor configuração e nem convergiu mais rápido, pois as tabelas 5.3 e 5.4 possuem o mesmo número de linhas, indicando que foi necessário o mesmo número de passos para executar por completo a heurística. Mas ainda continua demonstrando que, independente da aplicação, pelo estudo da área, chegamos a um valor, em reais, pelo menos igual ao da análise sem a área, mesmo sem possuir a menor função de custo.

Tabela 5.3: Resultado da Heurística sem análise da influência da área em sua avaliação, utilizando os resultados do Susan.

Config. Cache LI1	Config. Cache LD1	Config. Cache L2	Ciclos Totais	Energia Total (µJ)	Área Total (cm^2)	FC	Valor (R$)
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	26879128	13,4306049796614	0,172406	62,2391	2,324
(2048, 16, 1)	(2048, 16, 1)	(32768, 16, 1)	26911940	15,9729824310003	0,090145	38,7501	1,230
(2048, 16, 1)	(2048, 16, 1)	(65536, 32, 1)	26802805	14,5480441725444	0,121079	47,2121	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	26887160	14,0271477802331	0,121503	45,8249	2,324
(2048, 16, 1)	(4096, 16, 1)	(65536, 16, 1)	26853362	47,7239467934893	0,175778	225,2680	2,393
(2048, 16, 1)	(2048, 16, 2)	(65536, 16, 1)	26769408	17,7944809885212	0,174139	82,9507	2,324
(4096, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	26573220	13,5668866618642	0,175778	63,3708	2,393
(2048, 16, 2)	(2048, 16, 1)	(65536, 16, 1)	26148022	13,4435891822070	0,174139	61,2139	2,324

Tabela 5.4: Resultado da Heurística com análise da influência da área em sua avaliação, utilizando os resultados do Susan.

Config. Cache LI1	Config. Cache LD1	Config. Cache L2	Ciclos Totais	Energia Total (µJ)	Área Total (cm^2)	FC	Valor (R$)
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	26879128	13,4306049796614	0,172406	62,2391	2,324
(2048, 16, 1)	(2048, 16, 1)	(32768, 16, 1)	26911940	15,9729824310003	0,090145	38,7501	1,230
(2048, 16, 1)	(2048, 16, 1)	(65536, 32, 1)	26802805	14,5480441725444	0,121079	47,2121	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	26887160	14,0271477802331	0,121503	45,8249	2,324
(2048, 16, 1)	(4096, 16, 1)	(65536, 16, 1)	26853362	47,7239467934893	0,175778	225,2680	2,393
(2048, 16, 1)	(2048, 16, 2)	(65536, 16, 1)	26769408	17,7944809885212	0,174139	82,9507	2,324
(4096, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	26573220	13,5668866618642	0,175778	63,3708	2,393
(2048, 16, 2)	(2048, 16, 1)	(65536, 16, 1)	26148022	13,4435891822070	0,174139	61,2139	2,324

Na figura 5.3 temos os gráficos gerados dos resultados contidos nas tabelas 5.3 e 5.4, para melhor visualização desses. Sobrepondo-os podemos perceber que os pontos encontrados pela heurística, em ambos os casos, são exatamente iguais.

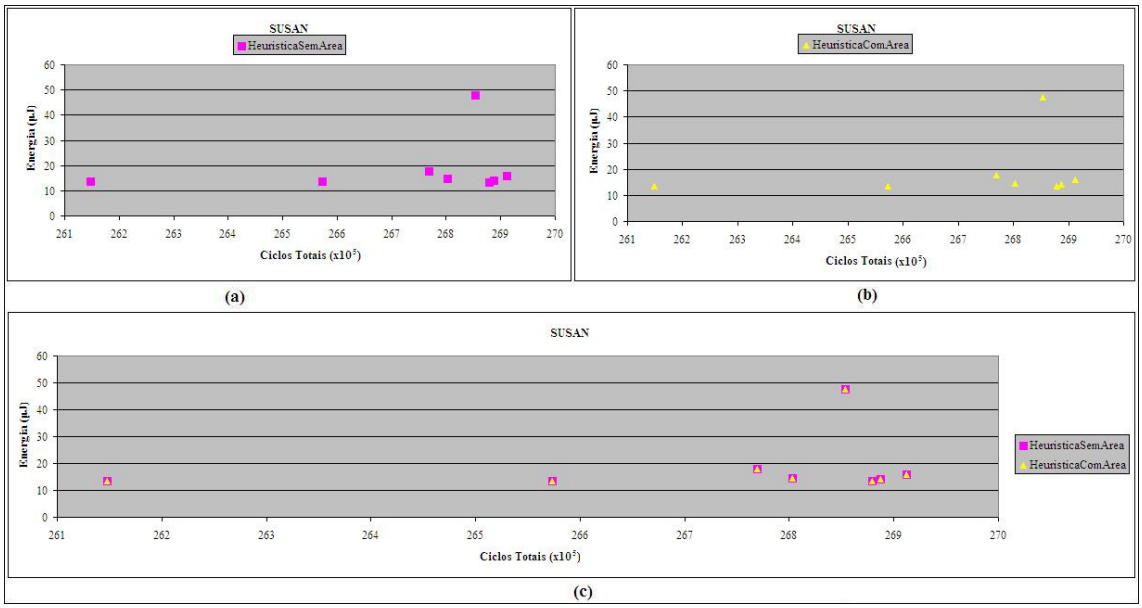

Figura 5.3: Gráficos das configurações selecionadas pela heurística TECH-CYCLES, referentes às tabelas 5.3 e 5.4, onde: (a) Aplicação da heurística sem análise de Área Total; (b) Aplicação da heurística com análise de Área Total; (c) Junção das duas análises.

O gráfico da figura 5.4 mostra a junção de todos os pontos gerados pela simulação das configurações pela aplicação Susan e as configurações selecionadas pela aplicação da heurística mostradas na figura 5.3.

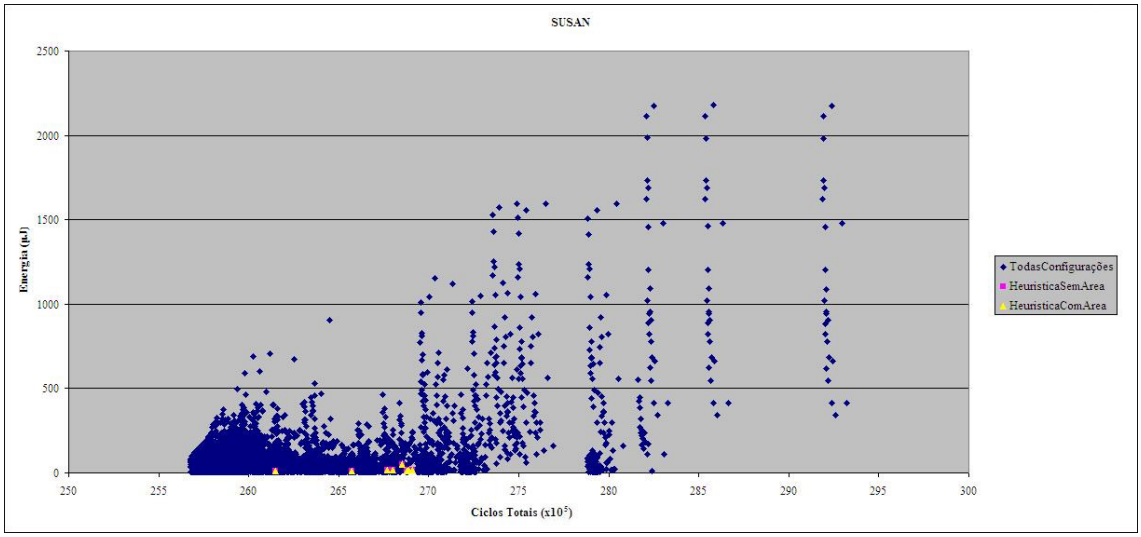

Figura 5.4: Representação gráfica das configurações obtidas do Susan em confrontamento com as obtidas pela heurística com análise da área e sem a análise da área.

O gráfico da figura 5.4 mostra, mais uma vez, a ineficiência da heurística TECH-CYCLES para este tipo de arquitetura, pois as configurações destacadas pela heurística ainda estão distantes dos pontos ótimos de consumo de energia e de número de ciclos totais.

Dando seguimento as análises dos dados, têm-se os resultados da execução da heurística na aplicação Dijkstra do *Benchmark*, observando nas tabelas 5.10 e 5.11 as configurações selecionadas pela heurística, para a análise da mesma sem considerar a área total e considerando a área total, respectivamente.

Ao comparar as tabelas 5.5 e 5.6, que representam os resultados da heurística, observa-se que algumas configurações encontradas, nesta aplicação, são diferentes, mesmo a heurística sendo a mesma. Porém, na execução dela com a análise de área em conjunto com a energia e os ciclos, volta-se a observar uma redução no número de linhas da tabela 5.6, indicando que a heurística convergiu mais rápido para um resultado final.

Apesar disso, pode-se verificar que a função de custo não é a menor dentre todas as configurações das tabelas, mas o valor em reais relacionado à construção da área da

configuração da memória cache analisada, é menor no caso da tabela 5.6, se comparada ao valor em reais selecionado na tabela 5.5. Sendo as configurações em destaque, nas tabelas 5.5 e 5.6, diferentes, observa-se que a inclusão da área na análise da execução da heurística influencia na seleção da melhor configuração, pois busca sempre manter o menor conjunto de valores para a área total, a energia consumida e o número de ciclos totais.

Tabela 5.5: Resultado da Heurística sem análise da influência da área em sua avaliação, utilizando os resultados do Dijkstra.

Config. Cache LI1	Config. Cache LD1	Config. Cache L2	Ciclos Totais	Energia Total (µJ)	Área Total (cm^2)	FC	Valor (R$)
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	95666698	587,9650973888930	0,172406	9697,6098	2,324
(2048, 16, 1)	(2048, 16, 1)	(32768, 16, 1)	106086569	1360,1086160497000	0,090145	13006,9550	1,230
(2048, 16, 1)	(2048, 16, 1)	(65536, 32, 1)	93012695	759,8270242741120	0,121079	8557,0839	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	91522701	260,8590036926030	0,121503	2900,8259	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	89986427	136,2681493806330	0,097749	1198,6260	2,324
(2048, 16, 1)	(4096, 16, 1)	(65536, 16, 4)	81209829	475,2596782437080	0,101121	3902,8416	2,393
(2048, 16, 1)	(2048, 16, 2)	(65536, 16, 4)	87093992	180,3261671604790	0,099482	1562,3972	2,324
(4096, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	82609170	135,5196942230680	0,101121	1132,0667	2,393
(8192, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	78910211	135,8604131245680	0,114086	1223,0902	2,529
(4096, 16, 2)	(2048, 16, 1)	(65536, 16, 4)	81712270	136,5210167492460	0,099736	1112,5992	2,393

Tabela 5.6: Resultado da Heurística com análise da influência da área em sua avaliação, utilizando os resultados do Dijkstra.

Config. Cache LI1	Config. Cache LD1	Config. Cache L2	Ciclos Totais	Energia Total (µJ)	Área Total (cm^2)	FC	Valor (R$)
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 1)	95666698	587,965097388893	0,172406	9697,6098	2,324
(2048, 16, 1)	(2048, 16, 1)	(32768, 16, 1)	106086569	1360,108616049700	0,090145	13006,9550	1,230
(2048, 16, 1)	(2048, 16, 1)	(65536, 32, 1)	93012695	759,827024274112	0,121079	8557,0839	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 2)	91522701	260,859003692603	0,121503	2900,8259	2,324
(2048, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	89986427	136,268149380633	0,097749	1198,6260	2,324
(2048, 16, 1)	(4096, 16, 1)	(65536, 16, 4)	81209829	475,259678243708	0,101121	3902,8416	2,393
(2048, 16, 1)	(2048, 16, 2)	(65536, 16, 4)	87093992	180,326167160479	0,099482	1562,3972	2,324
(4096, 16, 1)	(2048, 16, 1)	(65536, 16, 4)	82609170	135,519694223068	0,101121	1132,0667	2,393
(2048, 16, 2)	(2048, 16, 1)	(65536, 16, 4)	88952772	135,892238268995	0,099482	1202,5375	2,324

Na figura 5.5 temos os gráficos gerados para melhor visualizarmos os resultados da aplicação da heurística para as configurações de cache das tabelas 5.5 e a tabela 5.6.

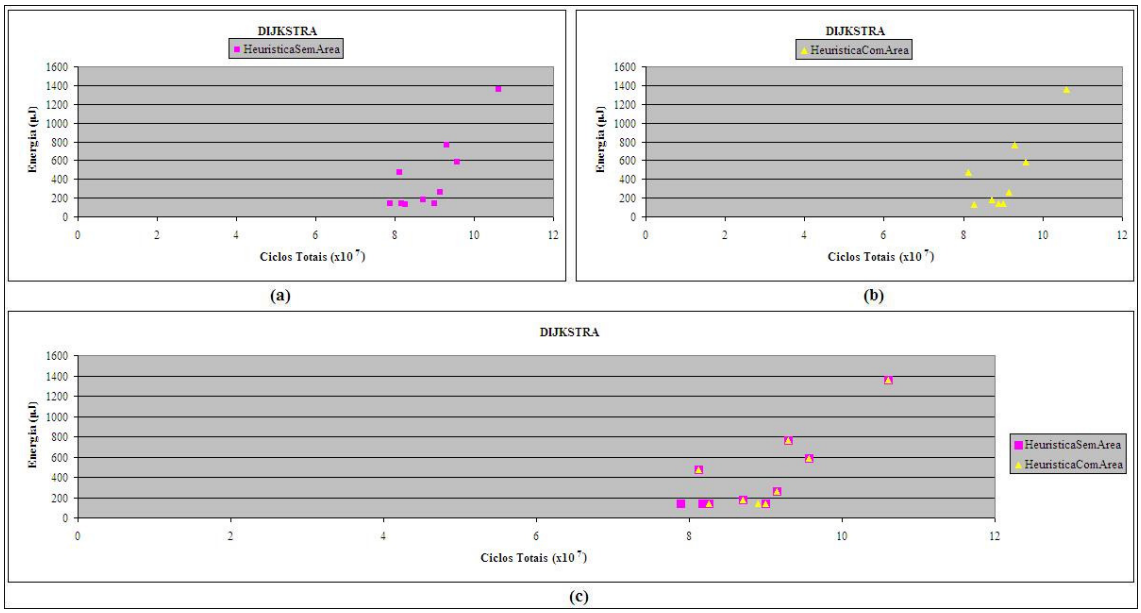

Figura 5.5: Gráficos das configurações selecionadas pela heurística TECH-CYCLES, referentes às tabelas 5.5 e 5.6, onde: (a) Aplicação da heurística sem análise de Área Total; (b) Aplicação da heurística com análise de Área Total; (c) Junção das duas análises.

O gráfico da figura 5.6 mostra a junção de todos os pontos gerados pelas configurações obtidas do Dijkstra e as configurações vindas da aplicação da heurística mostradas na figura 5.5, podendo-se observar, mais uma vez, a deficiência desta heurística para a arquitetura aplicada, pois os pontos ainda encontram-se longe dos pontos ótimos.

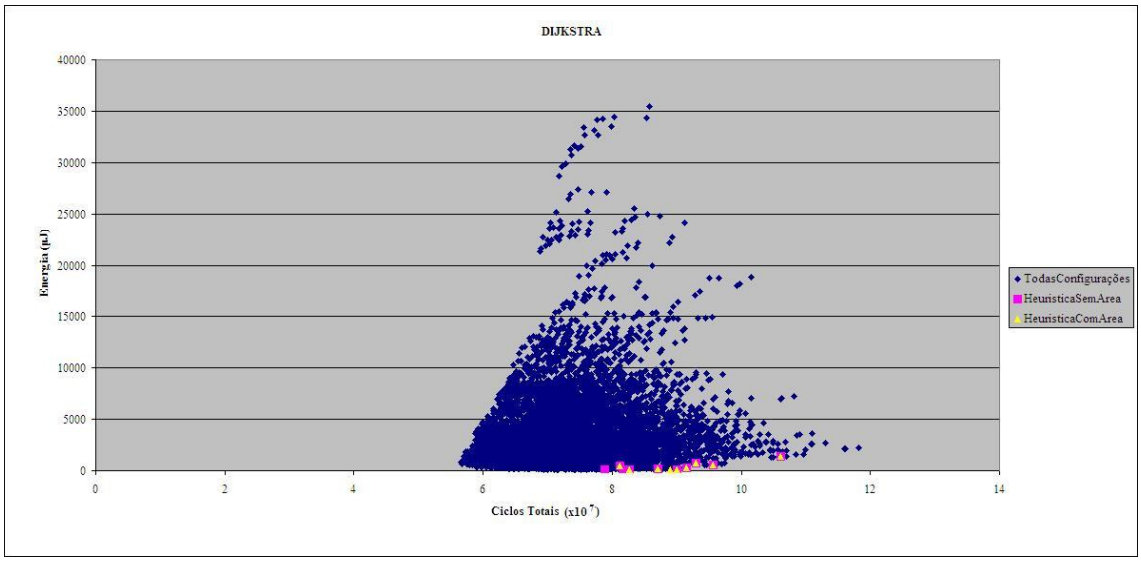

Figura 5.6: Representação gráfica das configurações obtidas do Dijkstra em confronta-mento com as obtidas pela heurística com análise da área e sem a análise da área.

Capítulo 6

Conclusões e Trabalhos Futuros

Um novo tipo de análise foi proposta neste trabalho de conclusão de curso, tendo como objetivo encontrar a melhor configuração dentre o vasto número de configurações existentes, sendo essa a que apresenta o menor valor de energia consumida, de ciclos totais e de área total, conjuntamente. Servindo como base de estudos para esse tipo de análise em memórias de dois níveis, com o segundo nível unificado. Sendo viável o estudo deste tipo de hierarquia tendo em vista o elevado espaço de exploração.

Foi realizada a incorporação da área na função de custo da heurística TECH-CYCLES, já existente, para que fosse possível uma verificação da combinação dessas 3 componentes. Assim como foram realizadas comparações entre a execução dessa heurística, em um primeiro momento analisando apenas os menores valores da energia total e do número de ciclos totais, e em seguida, a análise para essas duas componentes em conjunto com a área total. Sendo reduzido o espaço de exploração em aproximadamente 0,11% do número total de configurações geradas, e simuladas exaustivamente.

O uso de banco de dados para estes problemas de exploração facilitam bastante a manipulação das informações geradas, quer seja no armazenamento das informações, na unificação das informações do SimpleScalar com as do eCACTI, ou na busca e seleção das configurações durante a execução da heurística.

Foram utilizadas 3 aplicações do *Mibench Benchmark Suite*, para a validação dessa nova análise, o que se mostrou eficaz quando o foco principal for a construção do ASIC. Pois os valores para a energia total e para o número de ciclos totais nem sempre são os melhores dentre as configurações selecionadas pela heurística, mas tem-se

sempre o menor, ou pelo menos o mesmo, valor em reais ao comparar as duas análises, com e sem a área total.

Como trabalho futuro, fica, principalmente, a elaboração de uma heurística específica para a arquitetura de cache de dois níveis com o segundo nível unificado, analisando as três componentes bases desse trabalho: a energia total, o número de ciclos totais e a área total. Podendo ser desenvolvida utilizando-se dos princípios de banco de dados, assim como pode-se também utilizar os conhecimentos de redes neurais para que seja possível, através de mais de uma interação no campo amostral das configurações, conseguir chegar mais próximo da configuração ideal sem muito esforço, como ocorre na busca exaustiva por todas as configurações existentes.

Além da heurística, observa-se que os processadores atuais estão implementando, cada vez mais, mais níveis de cache, o que já se encontra no quarto nível. Uma outra proposta para trabalhos futuros é o aprofundamento deste estudo da contribuição da área para estas novas arquiteturas de processadores, para que se possa auxiliar na criação de suas futuras gerações, uma vez que, quanto mais níveis de cache existirem, maiores serão as áreas ocupadas, dependendo da tecnologia utilizada, claro.

Bibliografia

[1] Stallings, Willian. Arquitetura e Organização de Computadores. 5 ª Edição. Editora Prentice Hall, 2002. 792 p.

[2] Guia do Hardware – Como o Cache de Memória Funciona. Disponível em: http://www.clubedohardware.com.br/artigos/1410/1. Acessado em: 17/03/2008.

[3] Guilhermino, Abel. Uma Metodologia Para Exploração De Configurações De Memória Cache Visando Redução Do Consumo De Energia. 2006. 157 f. Tese de Doutorado em Ciências da Computação. Universidade Federal de Pernambuco. Recife.

[4] Hennessay, John L. e Patterson, David A. Organização e Projeto de Computadores – A interface Hardware/Software. 3ª Edição. Editora Campus, 2005, 512 p.

[5] http://www.newton.dep.anl.gov/askasci/comp99/CS017.htm, site apoiado pelo Departamento de Energia dos Estados Unidos da América, acessado em 30/05/2008.

[6] http://www.forumpcs.com.br/coluna.php?b=105004, acessado em 27/04/2008.

[7] Torres, Gabriel. Hardware - Curso completo, 4ª edição. Editora Axcel Books, 2001, 1468p.

[8] www.joaoantonio.com.br, acessado em 27/04/2008.

[9] http://pt.wikipedia.org/wiki/Mem%C3%B3ria_(computador), acessado em 27/04/2008.

[10] http://www.di.ufpb.br/raimundo/Hierarquia/Hierarquia.html, acessado em 27/04/2008.

ANEXOS

ANEXO 1

Arquivo contendo resultados da simulação do SimpleScalar

```
sim-cache: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.All
Rights Reserved. This version of SimpleScalar is licensed for academicnon-
commercial use.  No portion of this work may be used by any commercialentity,
or for any commercial purpose, without the prior written permissionof
SimpleScalar, LLC (info@simplescalar.com).

sim: command line: /home/washington/tcc/simplescalar/simplesim-3.0/sim-cache -
redir:sim /home/washington/tcc/simulacao/configuracao/conf001_qsort_small.txt
-cache:il1 il1:128:16:1:r -cache:dl1 dl1:128:16:1:r -cache:il2 dl2 -cache:dl2
ul2:1024:16:1:r /home/washington/tcc/bitcnts 75000

sim: simulation started @ Mon Mar 31 23:14:17 2008, options follow:

sim-cache: This simulator implements a functional cache simulator.  Cache
statistics are generated for a user-selected cache and TLB configuration,which
may include up to two levels of instruction and data cache (with any levels
unified), and one level of instruction and data TLBs.  No timing information
is generated.

# -config                     # load configuration from a file
# -dumpconfig                 # dump configuration to a file
# -h                    false # print help message
# -v                    false # verbose operation
# -d                    false # enable debug message
# -i                    false # start in Dlite debugger
-seed                       1 # random number generator seed (0 for timer
seed)
# -q                    false # initialize and terminate immediately
# -chkpt                <null> # restore EIO trace execution from <fname>
# -redir:sim
/home/washington/tcc/simulacao/configuracao/conf001_qsort_small.txt # redirect
simulator output to file (non-interactive only)
# -redir:prog           <null> # redirect simulated program output to file
-nice                        0 # simulator scheduling priority
-max:inst                    0 # maximum number of inst's to execute
-cache:dl1       dl1:128:16:1:r # l1 data cache config, i.e., {<config>|none}
-cache:dl2       ul2:1024:16:1:r # l2 data cache config, i.e., {<config>|none}
-cache:il1       il1:128:16:1:r # l1 inst cache config, i.e.,
{<config>|dl1|dl2|none}
-cache:il2                  dl2 # l2 instruction cache config, i.e.,
{<config>|dl2|none}
-tlb:itlb        itlb:16:4096:4:l # instruction TLB config, i.e.,
{<config>|none}
-tlb:dtlb        dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-flush                 false # flush caches on system calls
-cache:icompress       false # convert 64-bit inst addresses to 32-bit inst
equivalents
# -pcstat               <null> # profile stat(s) against text addr's (mult uses
ok)

  The cache config parameter <config> has the following format:

    <name>:<nsets>:<bsize>:<assoc>:<repl>

    <name>   - name of the cache being defined
    <nsets>  - number of sets in the cache
    <bsize>  - block size of the cache
    <assoc>  - associativity of the cache
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

    Examples:   -cache:dl1 dl1:4096:32:1:l
```

```
                -dtlb dtlb:128:4096:32:r

  Cache levels can be unified by pointing a level of the instruction cache
  hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache
  configuration arguments.  Most sensible combinations are supported, e.g.,

    A unified l2 cache (il2 is pointed at dl2):
      -cache:il1 il1:128:64:1:l -cache:il2 dl2
      -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

    Or, a fully unified cache hierarchy (il1 pointed at dl1):
      -cache:il1 dl1
      -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l



sim: ** starting functional simulation w/ caches **

sim: ** simulation statistics **
sim_num_insn               43621288 # total number of instructions executed
sim_num_refs                5117974 # total number of loads and stores
executed
sim_elapsed_time                  9 # total simulation time in seconds
sim_inst_rate          4846809.7778 # simulation speed (in insts/sec)
il1.accesses               43621288 # total number of accesses
il1.hits                   43600437 # total number of hits
il1.misses                    20851 # total number of misses
il1.replacements              20723 # total number of replacements
il1.writebacks                    0 # total number of writebacks
il1.invalidations                 0 # total number of invalidations
il1.miss_rate                0.0005 # miss rate (i.e., misses/ref)
il1.repl_rate                0.0005 # replacement rate (i.e., repls/ref)
il1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses                5119781 # total number of accesses
dl1.hits                    5118107 # total number of hits
dl1.misses                     1674 # total number of misses
dl1.replacements               1546 # total number of replacements
dl1.writebacks                 1173 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0003 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0003 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0002 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses                  23698 # total number of accesses
ul2.hits                      12772 # total number of hits
ul2.misses                    10926 # total number of misses
ul2.replacements               9906 # total number of replacements
ul2.writebacks                  891 # total number of writebacks
ul2.invalidations                 0 # total number of invalidations
ul2.miss_rate                0.4611 # miss rate (i.e., misses/ref)
ul2.repl_rate                0.4180 # replacement rate (i.e., repls/ref)
ul2.wb_rate                  0.0376 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
itlb.accesses              43621288 # total number of accesses
itlb.hits                  43621268 # total number of hits
itlb.misses                      20 # total number of misses
itlb.replacements                 0 # total number of replacements
itlb.writebacks                   0 # total number of writebacks
itlb.invalidations                0 # total number of invalidations
itlb.miss_rate               0.0000 # miss rate (i.e., misses/ref)
itlb.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
dtlb.accesses               5119781 # total number of accesses
dtlb.hits                   5119772 # total number of hits
```

```
dtlb.misses                      9 # total number of misses
dtlb.replacements                0 # total number of replacements
dtlb.writebacks                  0 # total number of writebacks
dtlb.invalidations               0 # total number of invalidations
dtlb.miss_rate              0.0000 # miss rate (i.e., misses/ref)
dtlb.repl_rate              0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                 81680 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                  9472 # program init'ed `.data' and uninit'ed
`.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest
address in stack)
ld_stack_size                16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian             0 # target executable endian-ness, non-zero
if big endian
mem.page_count                  29 # total number of pages allocated
mem.page_mem                  116k # total size of memory pages allocated
mem.ptab_misses                 30 # total first level page table misses
mem.ptab_accesses         185232395 # total page table accesses
mem.ptab_miss_rate          0.0000 # first level page table miss rate
```

ANEXO 2

Arquivo contendo resultados do eCACTI

```
Command line: /home/washington/tcc/eCACTI/eCACTI 2048 16 1 0.180000 1
paramFile
--------------------------------
---------- eCACTI 1.0 ----------
--------------------------------
Cache Subarray Parameters (C, B, A): (2048 B, 16 B, DM)
Number of Subarrays: 1
Ports (RW, R, W): (1, 0, 0)
Technology: 0.18 um, Vdd: 2.0 V

#Cache configurations expressed in (Ndwl, Nspd, Ndbl, Ntwl, Ntspd,
Ntbl) format

Area efficient configuration: (8, 1, 2, 1, 16, 1)
  Area = 12.785048, Time = 2.478120 ns, Power (ReadOp(dyn,lkg),
WriteOp(dyn,lkg)) = 1035.68 (1021.83876 13.83684) mW; 1023.80
(1020.68869 3.11317) mW;

Optimal time configuration: (1, 1, 16, 8, 4, 2)
  Area = 21.355334, Time = 0.828116 ns, Power (ReadOp(dyn,lkg),
WriteOp(dyn,lkg)) = 136.07 (135.74383 0.32653) mW; 100.35 (100.27760
0.07341) mW;

Lowest power configuration: (2, 1, 2, 1, 1, 4)
  Area = 51.097372, Time = 1.489929 ns, Power (ReadOp(dyn,lkg),
WriteOp(dyn,lkg)) = 42.18 (42.14213 0.03376) mW; 41.37 (41.36534
0.00750) mW;

Optimal Power-AccessTime-Area Config: (1, 1, 1, 2, 1, 4)

Power Stats:
------------
Read hit power: 57.3627 mw
Write hit power: 68.5336 mw

Read miss power: 121.5099 mw
Write miss power: 72.2157 mw

Timing Stats:
-------------
Access time: 1.13913 ns
Cycle time (wave pipelined):  0.535028 ns

Area Stats:
-----------
Aspect ratio (height/width): 1.036352
Total area one subarray : 0.003677 cm^2

Power Components (dyn, lkg mW):
--------------------------------

#NOTE: The power values are for a cache read hit

 Total Power all Banks :   57.33, 0.02833
```

```
  Data Array Split:
  -----------------
        decode :  0.9153, 0.0009507
      wordline :  0.4303, 2.25e-06
       bitline :    32.1, 0.01769
     sense_amp :   1.815, 0.0009855
  read_control :     3.2, 6.168e-05
 write_control :       0,       0

  Tag Array Split:
  ----------------
        decode :   2.013, 0.002339
      wordline :  0.1058, 9.322e-06
       bitline :   1.649, 0.002488
     sense_amp :   2.192, 0.002878
  read_control :  0.9924, 1.913e-05
 write_control :       0,       0
     tag comparison :   0.802, 0.0001196
valid signal driver :  0.3071,       0
 data output driver :   10.81,  0.0007788

Time Components:
----------------
 data side (with Output driver) (ns): 1.04089
 tag side (with Output driver) (ns): 1.13913
 subbank address routing delay (ns): 0
 decode_data (ns): 0.257347
 wordline and bitline data (ns) : 0.47633
 sense_amp_data (ns): 0.18675
 decode_tag (ns): 0.535028
 wordline and bitline tag (ns): 0.178705
 sense_amp_tag (ns): 0.11475
 compare time (ns): 0.252883
 valid signal driver (ns): 0.0577649
 data output driver (ns): 0.120463
 total_out_driver (ns): 0
 total data path (without output driver) (ns): 0.920427
 total tag path is dm (ns): 1.08137

Area Components:
----------------
 Aspect Ratio Total height/width: 1.036352
 Data array (cm^2): 0.002549
 Data predecode (cm^2): 0.000083
 Data colmux predecode (cm^2): 0.000019
 Data colmux post decode (cm^2): 0.000000
 Data write signal (cm^2): 0.000001
 Tag array (cm^2): 0.000939
 Tag predecode (cm^2): 0.000038
 Tag colmux predecode (cm^2): 0.000019
 Tag colmux post decode (cm^2): 0.000001
 Tag output driver decode (cm^2): 0.000026
 Tag output driver enable signals (cm^2): 0.000001
 Percentage of data ramcells alone of total area: 47.698319 %
 Percentage of tag ramcells alone of total area: 11.924580 %
 Percentage of total control/routing alone of total area: 40.377102 %
 Subbank Efficiency : 59.622898
 Total Efficiency : 49.824600
 Total area One Subbank (cm^2): 0.003677
 Total area subbanked (cm^2): 0.004401
```